UNIVERSITÉ DE
SHERBROOKE
Département du Génie Électrique & Informatique

Extension du protocole SIP pour le contrôle de la domotique distante

Mémoire de maîtrise ès sciences appliquées

Spécialité : génie informatique

Abdessamad CHTATOU

Sherbrooke (Québec) Canada Mai 2005

RÉSUMÉ

Il existe de nombreux standards destinés à contrôler les dispositifs servant à automatiser un habitat. En revanche, la plupart d'entre eux ne supportent pas le contrôle à distance de ces dispositifs, ni l'interconnexion de différentes technologies réseautiques. Un tel support devrait produire une radicale révolution dans le domaine de l'habitat communicant, puisqu'il serait possible de faire cohabiter, à la fois et dans le même domaine, non seulement les terminaux informatiques comme les PCs portables et les PDAs, mais également les dispositifs qui ont des tâches spéciales (une machine à café par exemple). Ainsi, l'usager final profiterait d'un menu riche contenant une multitude de services qui étaient inimaginables il n'y a pas longtemps.

Le protocole SIP (*Session Initiation Protocole*) qui est utilisé originellement pour la signalisation en téléphonie IP, peut fournir l'habilité à se connecter via Internet aux dispositifs domestiques. Ce projet propose une extension de SIP dans le cadre de la domotique distante : contrôle et gestion d'une résidence à distance. Une méthode "DO" pour contrôler les appareils domestiques ainsi que le support du protocole de description "DMP" (*Device Message Protocol*) ont été ajoutés au protocole SIP pour permettre l'envoi de commandes à des dispositifs dans la résidence (*Network Appliances*). L'ensemble de commandes a été testé avec une mise en œuvre sur la plateforme *OSGi*^{TM} et en utilisant le protocole *UPnP* sur le réseau résidentiel.

REMERCIEMENTS

Je tiens à remercier Mr. Philippe Mabilleau, Mon directeur de recherche, d'abord pour m'avoir accepté en tant qu'étudiant gradué en maîtrise dans son groupe de recherche, ensuite pour m'avoir proposé un sujet aussi important et finalement pour son encouragement et ses conseils qui étaient toujours une source d'inspiration pour moi, ainsi que pour son incroyable disponibilité.

Je remercie également Mr. Roch Lefebvre, professeur en département GEGI pour son soutien et ses encouragements durant ma maîtrise.

J'aimerais remercier aussi toute personne qui m'a aidé à dépasser toute difficulté de réalisation notamment :

- Le groupe de recherche de Telcordia©, en particulier Stan Moyer et Simon Tsang
- Ghazenfer Mansoor, membre du forum des développeurs de Prosyst©
- Jan Lucenius ex-membre du VTT, le centre de recherche technique de la Finlande
- Denis Vergnes et les autres étudiants du laboratoire DOMUS pour m'avoir aidé lors de mon séminaire de recherche
- Toute personne que j'ai oubliée et qui a contribué de près ou de loin à la réalisation de mon projet.

Abdessamad Chtatou

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION

Au sein de toute société industrielle, l'évolution scientifique a rendu essentielle l'automatisation de la vie quotidienne, et en particulier l'environnement résidentiel [TMM04]. Depuis l'aspirateur pour la moquette, en passant par le four à micro-ondes, jusqu'au téléphone portable dans la poche, il est devenu inimaginable de vivre sans avoir besoin de ce genre d'appareils. En plus, si une amélioration est apportée à ces machines en les dotant d'une habilité à se connecter au réseau, il serait possible alors de leur fournir, à la fois, un accès à des ressources d'information massive et un contrôle distant via Internet, ce qui permet, non seulement d'améliorer leurs fonctionnalités d'origine, mais également d'en ajouter d'autres. Ainsi, le four à micro-ondes aurait la capacité de naviguer sur Internet pour, d'une part, mettre à jour sa base de données des recettes cuisinières les plus récentes, et d'autre part, améliorer son degré d'autonomie. L'aspirateur pourrait également gagner du temps en se rapportant à une agence de maintenance, et en lui communicant régulièrement, son rapport de fonctionnement quotidien. Le téléphone portable, quant à lui, pourrait faire de la messagerie instantanée sur Internet tout en agissant comme démarreur à distance d'une auto.

L'applicabilité de l'automatisation n'est pas restreinte au domaine résidentiel. En effet, l'accès distant à des appareils "grands consommateurs

d'énergie" par un service de distribution d'électricité[1] pourrait pallier à des problèmes de gestion de consommation électrique. Notamment le contrôle de taux de consommation surtout dans les heures critiques, pourrait aider non seulement à économiser l'énergie, mais en plus à optimiser un tel vital service.

Il est à noter que le domaine de l'automatisation des habitats et celui de l'industrie des systèmes de sécurité, sont les deux champs les plus importants pour implémenter les appareils connectés au réseau (NAs)[2], puisqu'une grande partie du marché de ces NAs y est destinée.

L'industrie automobile est également devenue un autre domaine où l'on profite davantage du déploiement des NAs. Les véhicules modernes sont de plus en plus équipés de systèmes autopilotés ou informatisés, tel que le freinage, la suspension, l'injection du combustible, ou même la direction assistée. En outre, l'habilité à établir des communications distantes avec de tels systèmes permet le déploiement du contrôle distant et sécurisé représentant ainsi une option avancée caractérisant le véhicule du futur.

Il existe plusieurs technologies qui gèrent la communication avec et entre les NAs. La plus simple de toutes est le protocole X10 [3], qui permet à un usager de contrôler ses appareils domestiques via un contrôleur X10 en leur envoyant des commandes *ON/OFF* ou pour changer leur niveau d'intensité. Viennent ensuite des technologies comportant des

[1]Hydro-Québec par exemple.

[2]NA : ou *Networked Appliance*, ce sigle a été choisi particulièrement par le groupe de recherche de TELCORDIA pour désigner tout appareil fournisseur d'un service quelconque, et avec lequel il est possible de communiquer via le réseau.

[3]www.x10.com

protocoles plus complexes comme Jini[4] par exemple, qui offrent beaucoup plus de possibilités pour la communication entre les NAs. En revanche, ces protocoles sont principalement conçus pour fonctionner dans un environnement résidentiel simple[5] et ne supportent pas les communications distantes via Internet. En plus, la plupart d'entre eux leur manquent des mécanismes nécessaires de sécurité.

Ce mémoire est organisé selon les trois principales parties suivantes :

I - la première partie présente une définition de la problématique et des critères liés à la communication et au contrôle de la domotique distante

II - la deuxième partie dresse l'état de l'art des technologies utilisées dans ce projet de recherche

III - la troisième partie, quant à elle, décrit les outils et les étapes de mise en œuvre et d'intégration des technologies décrites dans la deuxième partie.

Finalement, une conclusion de ce mémoire aura pour objet de faire le point, en bref, d'abord, sur ce qui a été réalisé lors de ce projet, et ensuite d'identifier les défis qui restent à relever dans ce domaine en particulier.

[4]www.jini.org
[5]Ou tout réseau local LAN

Première partie

Définition de la problématique

CHAPITRE 1

INTRODUCTION

L'Internet nouvelle génération est largement désigné pour faire évoluer l'utilité et le rôle que peuvent jouer des appareils reliés au réseau. Par exemple, un réfrigérateur pourra préparer un inventaire de la provision alimentaire, et renouveler la commande si nécessaire. Encore un réveil combinera les données de l'agenda, le temps, et l'état de la route pour déterminer le temps le plus adéquat pour effectuer une tâche externe. Il est clair que ces appareils ont besoin de communiquer entre eux, de telle façon qu'un réveil puisse allumer la lampe de la salle du bain par exemple, pourtant, le mécanisme de communication que ces appareils peuvent utiliser est loin d'être une évidence. Le rôle d'un protocole de signalisation en l'occurrence de SIP est de répondre aux différentes exigences nécessaires pour standardiser ces mécanismes, par l'offre d'un moyen de communication fiable et sécurisé, que ce soit au sein même du réseau résidentiel, entre les dispositifs internes ou bien entre ce réseau et le monde extérieur via une passerelle.

1.1 COMMUNICATION INTRA-HABITATION

Le premier avantage de l'utilisation de SIP pour les appareils liés au réseau, est qu'il offre une connectivité vers un réseau WAN [MMT01]. Il est possible qu'au sein d'un habitat, les appareils soient connectés en utilisant des technologies comme X10, Jini, etc. Il est possible également que SIP puisse être utilisé pour faire communiquer ces appareils d'une part, comme technologie de substitution pour les autres protocoles, et d'autre part, comme protocole qui pourrait coexister avec d'autres dans un même domaine.

La communication entre-appareils dans un même domaine, peut être subdivisée en deux problématiques majeures : *la localisation et l'identification* de l'appareil, et *la communication* avec celui-ci. Il y a de nombreux standards pour gérer la communication entre-appareils, dans un même domaine (*HAVI, VESA Home Networking, JINI et UPnP* sont quatre exemples évidents) et il y a d'autres explicitement dédiés à la localisation et l'identification des services comme *SLP*[1] par exemple.

Même en tenant compte de la prolifération des standards gérant la communication en interne d'un habitat, il est possible d'identifier des caractéristiques communes entre elles :

- Les règles qui permettent de gérer les appareils pendant leurs communications seront ou bien codées dans l'appareil lui-même, ou importées depuis les serveurs.
- Ces règles pourraient être écrites en terminologie spécifique

[1]SLP (*Service Location Protocol*)-http ://www.openslp.org/doc/rfc/rfc2608.txt

conventionnelle. Par exemple, *Envoyer un message à un contrôleur ayant une adresse IP 13.28.44.67, port 2357 ; Démarrer l'unité numéro 4595.* Cela permet de standardiser davantage la communication entre-appareils.

1.2 COMMUNICATION EXTRA-HABITATION

Le développement des techniques pour l'accès à la maison depuis l'extérieur n'a pas eu la même attention qu'a connue le problème de communication à l'intérieur de la maison. Il y a un certain nombre de facteurs complémentaires qui doivent être pris en considération quand on met en œuvre une communication vers l'extérieur, notamment :

- Sécurité : La communication à l'intérieur exploite un niveau de sécurité physique, qui est quasiment perdu quand un accès arbitraire depuis l'extérieur est permis.
- Authentification : L'entité essayant de pénétrer à l'intérieur, a besoin d'être bien identifiée avant de lui permettre d'accéder.
- Fiabilité : l'accès extra-maison est caractérisé par son aspect architectural très large (WAN). Cela dit qu'il y a plusieurs facteurs supplémentaires que l'on doit prendre en considération, et qui sont liés surtout à la communication entre l'habitat et le monde externe. Par exemple, quand cette communication est perdue, les systèmes internes doivent continuer indépendamment leur bon fonctionnement.
- Graduation : Il y a bien évidement plusieurs maisons.
- Indépendance du protocole : Bien qu'à l'intérieur d'une même maison, il est acceptable que différents protocoles soient utilisés pour la

communication entre-appareils, la communication avec le monde externe exige le déploiement des protocoles indépendants des systèmes internes, puisque les détails exacts des unités à l'intérieur de l'habitat ne pourraient pas être connus depuis le monde extérieur.

- Nomination et Localisation : Les terminaux à l'intérieur de la maison doivent être nommés et localisés depuis l'extérieur.

Nous venons de voir que la communication à distance avec les appareils domestiques passe par deux étapes. D'abord, l'accès à l'habitat où se trouvent ces appareils, ensuite, la communication avec les appareils qui passe généralement via d'autres dispositifs intermédiaires. Il s'agit d'une communication entre les dispositifs à l'intérieur de l'habitat.

Dans le prochain chapitre nous verrons comment cette communication distante est possible en utilisant un protocole de signalisation comme SIP. Nous verrons également l'utilité de SIP avec la passerelle de services **OSGi**^{TM}.

CHAPITRE 2

SIP ET OSGI : UTILITÉ

2.1 POURQUOI SIP ?

La réponse la plus évidente est que SIP répond à toutes les exigences pour accéder aux équipements depuis un réseau WAN et cela permet la réutilisation de l'infrastructure qui a été construite pour SIP dans le nouveau domaine. Différentes technologies ont été développées pour accéder aux équipements depuis le WAN, cependant, SIP était l'unique protocole que l'on a trouvé qui répond à toutes les exigences demandées[1]. Parmi ces exigences, mentionnons :

- Supporter un adressage et un nommage abstraits : les équipements domestiques situés derrière une passerelle, un pare-feu ou un *NAT*[2], peuvent ne pas être directement adressables, et leurs adresses peuvent ne pas être connues.
- Garantir la sécurité : l'authentification et la confidentialité. Le contrôle d'accès est nécessaire, soit par tâche, soit par terminal. De plus, afin de

[1] Il y a des protocoles, comme HTTP, qui répondent à quelques exigences, mais pas toutes

[2] *Network Address Translation* : Système de traduction d'adresses IP entre différents réseaux (en général, entre des réseaux locaux et Internet).

garder un niveau d'intimité désiré, le contenu des messages doit être privé afin de se préserver.

- Supporter les quatre mécanismes de communication suivants pour les terminaux :

mécanisme	**usage**	**exemple**
flux de données média	sessions	regarder des images internes
contrôle	messagerie instantanée	allumer la lumière externe
requêtes	état d'un équipement	quelle est la température interne
événement asynchrones	notification	avertir quand l'alarme tombe en panne

TAB. 2.1 – Mécanismes de communication pour terminaux

- Avoir l'habilité à transporter différents types MIME.
- S'interfacer avec diverses technologies du réseau résidentiel (p.e HAVi, UPnP, Jini, ...).
- Fournir un support pour la mobilité des appareils intelligents[3] que ce soit à l'intérieur du même domaine, dans celui du fournisseur de services, ou à travers plusieurs domaines du fournisseur.

En outre, l'utilisation de SIP pour la domotique permet la réutilisation d'une éventuelle infrastructure SIP préexistante (ex, pour la VoIP, la messagerie instantanée ...), et cela pour la totalité du nouvel ensemble de services.

[3]le mot intelligent désigne ici tout appareil contrôlable à distance par l'intermédiaire d'une ou plusieurs technologies citées plus haut.

2.2 PASSERELLES RÉSIDENTIELLES

La complexité liée à l'équipement de la maison du futur réside dans la variété des protocoles et standards utilisés, ainsi que dans les exigences des services fournis.

Le raccordement des appareils au réseau local peut être effectué selon différentes manières, principalement en utilisant un support métallique ou Hertzien. Comme chaque système possède des capacités différentes de vitesse, de distance, et de volume de données transportées, différents supports conviennent pour divers objectifs :

Ligne à haute tension : ceci se réfère à l'installation électrique existante dans l'habitat. Son avantage principal est la préexistence des prises électriques partout dans la maison. Cependant, ce genre de support ne permet pas de transporter un grand flux de données à cause de la capacité limitée du fil métallique. Les principaux protocoles permettant d'établir une communication par un tel média sont : $\widehat{\text{X10}}$, *CEBus*® , *LonWorks, HomePlug*® *, et EHS* .

Ligne téléphonique : ce genre de support peut acheminer des signaux d'une bande passante très large, comme la communication de la voie et les données. Le *HomePNA* est le majeur standard utilisé pour ce contexte.

Ligne Hertzienne : Il y a plusieurs solutions sans fil désignés pour tout type de réseau, de l'éclairage traditionnel, et réseau de sécurité, jusqu'à la communication des données, et quelques systèmes de divertissement. Les protocoles classés dans ce groupe comprennent *HomeRF, Bluetooth, et IrDA.*

Nouvelles installations électriques : ce sont des systèmes spécifiquement

conçus pour la communication des données, comprenant le RG-6, désigné pour les systèmes de divertissement multi-chambres, et le CAT 5 pour la communication des données, du son et de la vidéo. Les standards populaires de communication qui requièrent ce genre de médias sont, entre autres, *FireWire (IEEE 1394), USB, et EIB.*

Même s'il existe plusieurs protocoles, aucun d'eux ne peut satisfaire tous les besoins, à cause de la diversité des caractéristiques concernant le coût et le matériel.

La solution la plus naturelle pour combiner ces différents réseaux est un nœud central qui pourrait jouer le rôle d'un pont entre les différents réseaux locaux et l'Internet c'est là exactement le but d'une passerelle résidentielle.

Dans le but de pouvoir installer et remplacer des modules logiciels dynamiquement sur la passerelle, le logiciel doit être conçu comme des blocs de programmes séparés interagissant entre eux. Ces blocs de programmes peuvent être appelés des services, et une passerelle hébergeant un logiciel structuré ainsi, est appelée une passerelle de services.

2.3 OSGi^{TM} :PASSERELLE DE SERVICES CHOISIE

L'OSGi [oA03] est l'un des candidats[4] les plus convaincants comme passerelle de services. Cela est dû au fait qu'il est déjà assez mature, et permet aisément de construire des ponts vers d'autres standards.

La plateforme **OSGi**^{TM} est une structure sécurisée à objectif général, basée

[4]Il existe d'autres standards pour le contrôle de l'habitat distant comme : *CableHome*, *PacketCable*, *DSL Forum* et *UPNP* etc.

sur la technologie Java. Elle supporte le déploiement des applications de services extensibles et téléchargeables que l'on nomme *bundles* .

Sur la passerelle **OSGi**^{TM} il est possible d'importer et d'installer les paquetages, quand l'on en a besoin, et les désinstaller dans le cas contraire. Les modules peuvent être installés et désinstallés à la volée, sans devoir relancer tout le système. Cela entraîne la création d'une architecture très flexible offrant les possibilités suivantes :

- Le contrôle et les diagnostics à distance.
- La mise à jour dynamique du logiciel (i.e la gestion du cycle de vie).
- L'administration à distance.
- La construction des systèmes ouverts aux applications trois-tiers.

Il y a plusieurs domaines dans lesquels les passerelles de services répondant aux derniers critères sont très utiles, entre autres :

- L'habitat.
- L'automatisation du bureau et du bâtiment.
- L'automatisation industrielle.
- Le soin médical.

Afin de donner une solution innovatrice dans le domaine des passerelles résidentielles[5], nous allons ajouter aux options[6] qu'offre **OSGi**^{TM} la force d'un protocole de communication comme SIP, en installant un proxy SIP, et en déployant des services SIP dans un environnement résidentiel géré par **OSGi**^{TM}. Cette adjonction va permettre ce qui suit :

[5]Le domaine des passerelles résidentielles est considéré comme un champs particulier des applications de la plateforme des services **OSGi**^{TM}.

[6]Outre les services que l'on crée et que l'on installe au besoin sur l'**OSGi**^{TM}, celui-ci est livré avec des services de base comme le *HTTP*

- Spécifier des règles supplémentaires de communication entre l'habitat et le monde externe.
- Renforcer l'aspect sécuritaire, en ajoutant une autre couche d'abstraction.
- Donner des identifications standardisées aux différents appareils connectés au réseau interne.

SIP est un protocole de la téléphonie sur IP utilisé dans ce projet pour communiquer avec les appareils domestiques. Ses nombreuses caractéristiques ont en fait un protocole bien placé pour le contrôle des dispositifs à distance. Plus loin dans ce mémoire nous verrons, plus en détails, pourquoi SIP, **UPnP**^{TM} et **OSGi**^{TM} sont étudiés par rapport au problème posé. La prochaine partie, quant à elle, fait le survol sur quelques technologies liées au domaine de la domotique. L'accent sera mis, généralement, sur la téléphonie sur IP, surtout sur SIP, mais aussi sur les passerelles de service basées sur **OSGi**^{TM}. Finalement, nous décrivons les technologies qui accompagnent SIP, dans ce projet, pour le contrôle distant de la domotique, à savoir **UPnP**^{TM} et $\widehat{\text{X10}}$.

Deuxième partie

État de l'art

CHAPITRE 3

PROTOCOLE SIP

3.1 VOIX ET MULTIMÉDIA SUR IP

Jusqu'au milieu des années 90, les organismes de normalisation ont tenté de transmettre les données de manière toujours plus efficace sur des réseaux conçus pour la téléphonie. À partir de cette date, il y a eu un changement de paradigme. C'est sur les réseaux de données, en particulier sur l'Internet, que l'on s'est évertué à convoyer la parole. Il a donc fallu développer des algorithmes de codage audio plus fiables et introduire des mécanismes de contrôle de la qualité de service dans les échanges audio/vidéo. Ces communications multimédia étant possibles sur divers types de réseaux, il s'est avéré nécessaire de les standardiser afin d'assurer la compatibilité entre applications. L'UIT (Union Internationale des Télécommunications) et l'IETF (*Internet Engineering Task Force*) ont élaboré des familles de standards regroupés sous les appellations génériques H.32x, SIP (*Session Initiation Protocol*) et MGCP (*Media Gateway Control Protocol*).

3.2 INTRODUCTION À SIP

Session Initiation Protocol (SIP) est un standard de l'IETF pour organiser des téléconférences via IP [RSC+02]. SIP est un protocole de contrôle basé sur ASCII, qui est utilisé pour établir, maintenir, et terminer les appels entre deux ou plusieurs nœuds. Comme la plupart des protocoles VoIP, SIP est désigné pour les fonctions de signalisation et de contrôle de session. Ces fonctions sont encapsulées dans un paquet et envoyées dans le réseau téléphonique. La fonction signalisation permet à l'information communiquée d'être transmise à travers les frontières du réseau, tandis que le contrôle de session offre le pouvoir de gérer les attributs d'un appel de bout-en-bout.

SIP fournit les possibilités de :

- Localiser un nœud distant : SIP supporte la résolution des adresses, l'enregistrement des noms, et la redirection des appels.
- Déterminer les caractéristiques médias du nœud distant : Via le *Session Description Protocol* (SDP). SIP détermine le "plus bas niveau" de services communs entre les deux nœuds en communication. Les conférences sont établies en utilisant les capacités médias qui peuvent être supportées par toutes les extrémités.
- Vérifier la disponibilité du nœud distant : si un appel ne peut pas être établi à cause de la non-disponibilité du nœud distant, SIP détermine si la partie appelée est déjà sur la ligne ou si elle n'a pas répondu au nombre déterminée de sonneries. Il rend ainsi un message indiquant la cause pour laquelle l'autre partie n'est pas disponible.
- établir une session entre les deux parties, appelante et cible : Si l'appel

peut être établi, SIP établit une session entre les deux extrémités. SIP supporte également les changements durant l'appel, comme l'ajout d'une autre partie à la conférence, ou le changement des caractéristiques du support.

- Manipuler la transmission et la terminaison des appels : SIP gère le transfert des appels d'un nœud à l'autre. Durant la transmission, SIP établit simplement une session entre la partie cessionnaire et le nouveau nœud (spécifié par la partie source) et termine la session entre la cessionnaire et la partie appelante. À la fin d'un appel, SIP termine les sessions entre toutes les parties.

Côté architecture, SIP est basé sur plusieurs protocoles de transport [Fig.3.2], notamment :

RSVP : (*Resource Reservation Protocol*) pour réserver les ressources réseaux sur IP avec une excellente qualité de service (QoS)

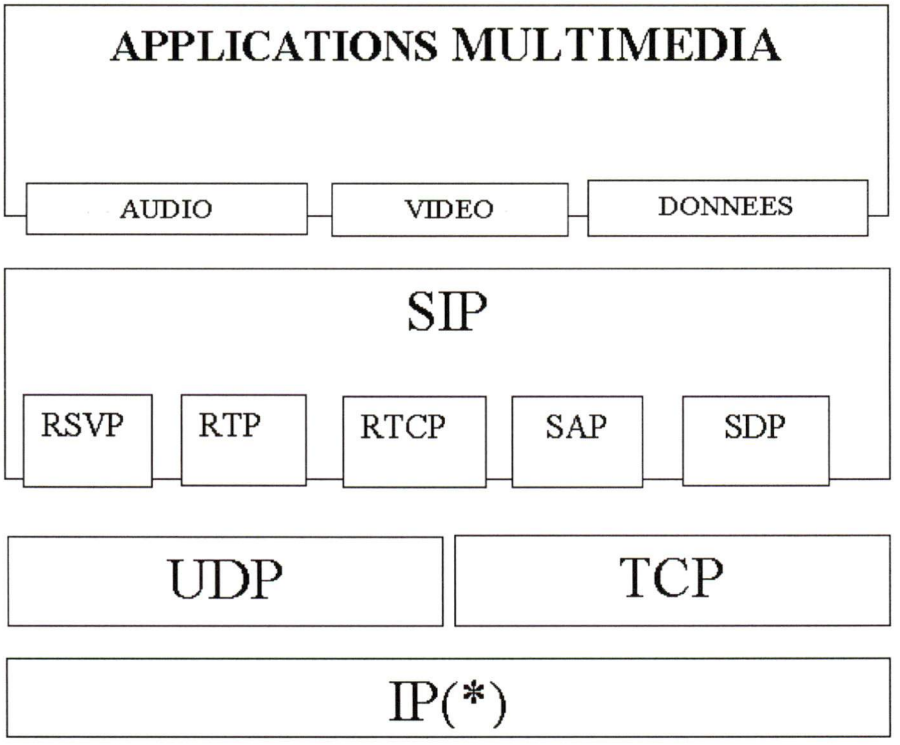
RTP : (*Real-time Transport Protocol*) pour transporter des informations en temps réel avec une excellente qualité de service

RTCP : (*Real-Time streaming Control Protocol*) pour assurer le contrôle de flux de données multimédia

SAP : (*Session Announcement Protocol*) pour préciser si les sessions mutimedia ouvertes le sont en multicast

SDP : (*Session Description Protocol*) est un protocole de description des sessions multimédia.

SIP peut également supporter les deux modes de connexion, fiable et non-fiable (*TCP/UDP*).



(*)SIP peut être également utilisé sur ATM(AAL5), X25 et "*frame relay*".

FIG. 3.1 – Architecture en couches de SIP

3.3 COMPOSANTES DE SIP

SIP est un protocole *peer-to-peer*. Les pairs dans une session s'appellent *User Agents* (UAs). Un agent utilisateur peut fonctionner selon l'un des deux rôles suivants :

- Agent Utilisateur Client (UAC) : une application cliente qui déclenche une requête SIP
- Agent Utilisateur Serveur (UAS) : une application serveur qui contacte l'utilisateur quand elle reçoit une requête SIP de celui-ci et lui rend une réponse par la suite.

Typiquement, un utilisateur SIP peut être à la fois un UAC ou UAS, mais il ne peut jouer qu'un seul rôle parmi les deux par transaction.
D'un point de vue architectural, les composantes physiques d'un réseau SIP peuvent être regroupées selon deux catégories : clients et serveurs.

3.3.1 Clients SIP

Les clients SIP comprennent :

- Téléphones : peuvent se comporter soit comme UAS ou UAC. Les *Softphones*[1] sont capables d'initier une requête SIP et répondre au requêtes reçues
- Passerelles : fournissent un contrôle d'appel. Elles offrent divers services, le plus commun est celui de la fonction de transfert entre les parties d'une téléconférence SIP et d'autres types de terminaux. Cette fonction comporte la coordination entre les formats de transmission et les procédures de communication.

3.3.2 Serveurs SIP

Les serveurs SIP comprennent :

- *Proxy server* (PS) : C'est l'unité intermédiaire qui reçoit une requête SIP et la redirige vers le destinataire approprié. En général, les PS reçoivent les messages SIP et les renvoient vers le prochain serveur SIP sur le réseau. Les PS peuvent fournir plusieurs autres fonctions, notamment, l'authentification, le contrôle d'accès, le routage, la retransmission fiable

[1]Softphones : logiciels permettant de contrôler des dispositifs téléphoniques

des requêtes, et la sécurité.

- *Redirect Server* (RS) : fournit au client les informations relatives au prochains nœuds que le message doit traverser. Ainsi le client pourrait contacter le dit nœud ou l'UAS directement.
- *Registrar server* (RG) : traite les requêtes des UACs pour l'enregistrement de leurs emplacements en cours.

3.4 Comment fonctionne SIP ?

SIP est un protocole simple basé sur le format ASCII. Il utilise requêtes et réponses pour établir la communication entre divers composants sur le réseau afin de préparer une conférence entre deux ou plusieurs clients.

Les utilisateurs SIP sont identifiés par des adresses SIP uniques. Une adresse SIP est similaire à une adresse e-mail, et elle est de la forme de : "*sip:userID@gateway.com*". Le *userID* peut être, ou bien un nom d'utilisateur, ou une adresse E.164[2].

Les utilisateurs s'enregistrent depuis le RS en utilisant leurs adresses SIP privées. Par la suite, le RS fournit ses informations au serveur local sur demande.

Quand un utilisateur lance un appel, une requête SIP est envoyée à un serveur SIP, elle comprend à la fois l'adresse de l'appelant (dans l'en-tête *From*), ainsi que l'adresse de l'appelé (dans l'en-tête *To*).

De temps à autre, un utilisateur pourrait se déplacer entre plusieurs nœuds. L'emplacement de l'utilisateur peut être dynamiquement enregistré par le serveur SIP. Celui-ci peut utiliser différents protocoles (entre autres : *finger, rwhois, LDAP...*) afin de localiser un client. Comme un utilisateur peut se connecter depuis plus d'une station, il peut avoir plus d'une adresse. Si la requête passe à travers un PS SIP, celui-ci va essayer chaque adresse retournée jusqu'à ce qu'il parvient à localiser l'utilisateur. Par contre, si cette requête passe par un RS SIP, ce dernier redirigera toutes les adresses

[2]Adresse E.164 : Une adresse composée de numéros et structurée comme un numéro de téléphone. En particulier, un numéro de téléphone est une adresse E.164. « E.164 » est le nom de la norme qui définit ces adresses. Exemple : 0123452389

à l'appelant dans le champs *Contact header* lié à la réponse. Le schéma 3.2 montre un exemple de session SIP établie entre deux utilisateurs : un client (UAC) et un serveur (UAS).

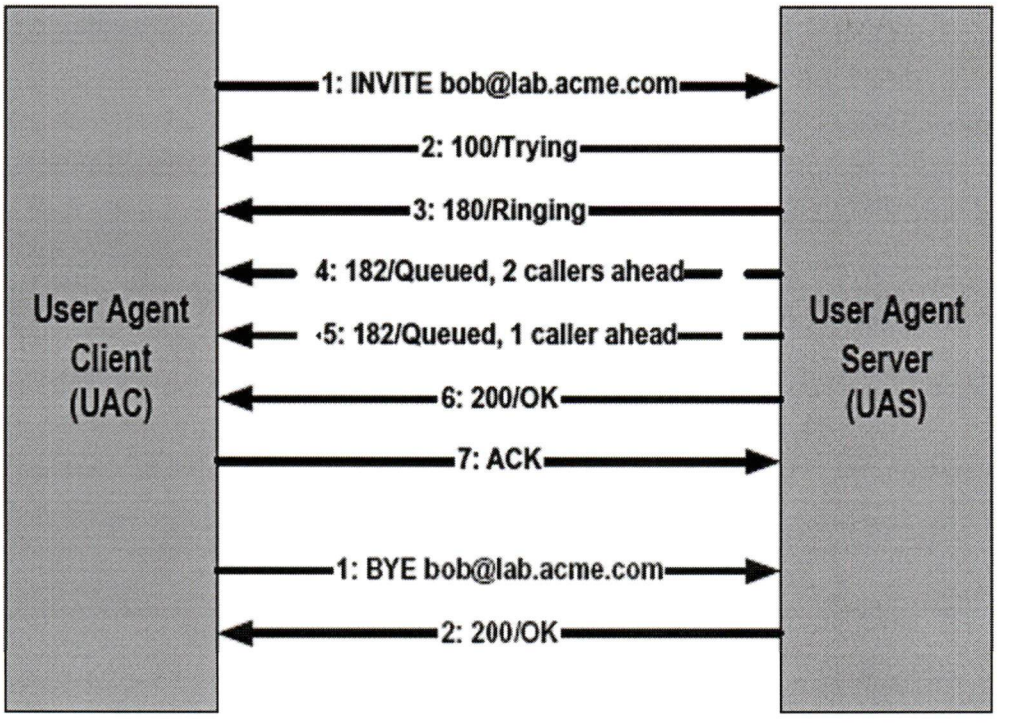


FIG. 3.2 – Exemple de session SIP entre deux utilisateurs

Description

1 L'appelant UAS envoie un message *INVITE* à l'adresse : sip:bob@acme.com. Ce message contient également un paquet SDP pour la description des capacités média relatives à l'appelant.

2 Le UAS reçoit la requête et répond tout de suite avec un message de type *100-Trying*.

3 Le UAS commence un *"ringing"* avisant *Bob* du nouvel appel et, en même temps, un message de type *180-Ringing* est transmis au UAC.

4 Le UAS envoie un message *182-Queued* pour informer que l'appel est dans la fille d'attente.

5 Même message avec un avancement de l'appel dans la file d'atente.

6 *Bob* prend l'appel et le UAS envoie un message *200-OK* à l'appelent. Ce même message comporte une description des capacités média relatives à l'appelé.

7 L'appelant UAC envoie un *ACK* pour confirmer que le *200-OK* a été reçu.

3.5 Comparaison entre SIP et H.323

Le protocole H.323 [Dat98] est apparu comme un standard multimédia de l'Union International de Télécommunication (ITU). Il est utilisé à la fois pour la téléphonie et la vidéo. Le H.323 incorpore divers protocoles, comme Q.931 pour la signalisation, H.245 pour la négociation et l'enregistrement d'état et l'enregistrement d'admission pour la gestion de session. H.323 était le premier standard pour le contrôle des appels sur VoIP. La première version de sa spécification a été approuvée en 1996.

SIP et H.323 ont été conçus pour les fonctions de contrôle de session et de signalisation dans une architecture de contrôle d'appels distribuée. Bien que les deux protocoles puissent être utilisés pour communiquer avec des nœuds moins intelligents, leur spécialité est d'établir des communications

avec des nœuds intelligents.

Dans ce projet SIP a été choisi par rapport à H.323 car SIP possède, en particulier, deux points forts [cf.Tab.3.1] qui sont favorables vis-à-vis du présent sujet. Mentionnons :

1 Son extensibilité : SIP permet le développement de sa pile, que ce soit pour changer l'une ou plusieurs de ses composantes ou pour y ajouter d'autres

2 la lisibilité de son encodage : ce dernier basé sur du texte, il est plus percevable par le développeur ce qui permet d'éditer sa spécification assez aisément.

Aspect	**SIP**	**H.323**
Clients	Intelligent	Intelligent
intelligence du réseau et des services	Présentée par les serveurs (PS,RS,RG)	Présentée par les *gatekeepers*
Modèle utilisé	Internet/Web	Téléphonie/Q.SIG
Protocole de signalisation	UDP ou TCP	TCP (UDP est optionnel dans la version 3)
Protocole support	RTP	RTP
Base d'encodage	ASCII	Binaire (codage ASN.1)
Autres protocoles utilisés	Protocoles IETF/IP, comme SDP, HTTP/1.1, IPmc, et MIME	Protocoles ITU / ISDN, comme H.225, H.245, et H.450
Interopérabilité	universel	universel
Complexité	adéquate	Haute (plusieurs protocoles utilisés : H.450, H.225.0, H.245)
Extensibilité	Ouvert à de nouvelles extensions	Difficile à décoder

TAB. 3.1 – Comparatif entre SIP et H.323

CHAPITRE 4

PLATEFORME **OSGi**^{TM}

L'**OSGi**^{TM} (*Open Services Gateway Initiative*) [oA03] a été fondé en mars 1999 dans l'objectif de fournir un forum pour le développement des spécifications ouvertes liées aux services livrables depuis l'Internet vers des réseaux et équipements locaux. Le forum permet également d'accélérer la demande des produits et des services dans le monde entier en investissant dans le marché et quelques programmes d'éducation pour l' utilisateur.

L'**OSGi**^{TM} compte actuellement 75 membres des secteurs tels que : fournisseurs d'Internet, opérateurs de réseau, matériel informatique, fabricants de logiciel, institutions académiques, organismes gouvernementaux et à but non lucratif et utilisateurs finaux. Parmi les établissements inscrits au forum **OSGi**^{TM} citons : *ABB, Schindler Aufzuge, Adtranz PUTCC, Sony, Philips, Compaq, IBM, Motorola, France Telecom, Ericsson, Toshiba, Sun* etc.

4.1 Mission d'OSGi^{TM}

Le but essentiel d'**OSGi**^{TM} est de définir et d'encourager l'adoption rapide de spécifications ouvertes pour la livraison des services contrôlés pour les réseaux domestiques, pour les véhicules et pour d'autres environnements.

Pour réaliser sa mission, l'**OSGi**^{TM} prévoit :

- L'organisation d'un forum et d'un environnement ouverts et indépendants pour ses membres afin, d'une part, d'approuver les spécifications initiales et d'adopter des révisions et des améliorations suggérées et, d'autre part, pour permettre aux utilisateurs de rencontrer des développeurs et des fournisseurs de produits et de services pour identifier les contraintes nécessaires à l'interopérabilité et à l'utilisabilité générale [oA03].
- La communication des spécifications aux agences et aux groupes spécialisés en vue de les standardiser.
- La sensibilisation des communautés productrices et consommatrices de la valeur et des avantages des produits et des services **OSGi**^{TM} via la publicité, les publications, les salons professionnels, les conférences et autres programmes.
- Le soutien de la création et de l'implémentation de procédures uniformes d'essais de conformité et de processus qui favorisent l'interopérabilité.
- Le maintien des rapports et des liaisons avec les établissements éducatifs, les instituts de recherche gouvernementaux, les consortiums de technologie et des organismes qui supportent et contribuent au développement des spécifications d'**OSGi**^{TM}.
- L'organisation de compétitions pour le développement de nouveaux

produits et services basés sur les spécifications d'**OSGi**^{TM} en conformité avec les lois et règlements anti-trust.

4.2 Pourquoi OSGi^{TM} ?

L'**OSGi**^{TM} influence et associe deux tendances clefs du marché, d'une part, l'ubiquité de la connectivité au réseau WAN, et l'accès Internet dans les maisons, les bureaux, les véhicules et les téléphones portatifs, et d'autre part, l'apparition de nouvelles applications, de services et d'équipements pour les réseaux.

Les terminaux "*passerelles*" peuvent être, un récepteur satellitaire numérique, un câble ou un modem DSL, un PC, un téléphone web ou une passerelle résidentielle.

Auparavant, les fournisseurs des services sont intervenus exclusivement dans leurs domaines respectifs, à savoir, un réseau téléphonique ou informatique, des services de câble, de divertissement, la gestion d'énergie ou la télématique. Avec l'apparition des services de la passerelle, les divergences entre ces diverses entités ont été réduites.

Actuellement de multiples services (par exemple, la téléphonie traditionnelle, l'accès Internet, la vidéo sur demande, les divertissements interactifs, le commerce mobile, la gestion de télémétrie et d'énergie, la commande d'appareils ménagers, etc) peuvent être fournis et contrôlés par n'importe quel fournisseur de services. Ces services sont offerts via Internet et les réseaux WAN, au bureau, à la maison et aux équipements portatifs.

Alors qu'il existe plusieurs réseaux grande distance et des normes de gestion

de réseaux domestiques, il n'y a aucune spécification pour la livraison d'un service. Les spécifications d'**OSGi**^{TM}, heureusement, fournissent la "*colle*" dans cette nouvelle chaîne de création de valeur, à travers une structure de plate-forme ouverte et indépendante et des API's qui permettent la livraison dynamique des services contrôlés[1] avec des métriques sécurisées, scalables et fiables.

4.3 PRODUITS BASÉS SUR OSGI^{TM}

Il existe plusieurs passerelles résidentielles se basant sur le noyau OSGi^{TM}. Chaque passerelle utilise les services fournis avec l'**OSGi**^{TM} [oA03] et y ajoute d'autres selon la vision et les objectifs du développeur de celle-ci. Dans cette section, quelques passerelles utilisant **OSGi**^{TM} sont présentées avec leurs caractéristiques respectives. Par la suite, les motivations derrières le choix de la passerelle utilisée dans ce projet sont évoquées.

4.3.1 JES

Le logiciel **Java Embedded Server**^{TM}**(JES)** est un serveur Java destiné au marché des réseaux résidentiels. Il s'agit d'une nouvelle classe de serveurs qui réside sur la passerelle **OSGi**^{TM} et facilite un contrôle sûr et dynamique du cycle de vie et de caractéristiques de chaque service, ainsi que le déploiement d'applications. Le contrôle du cycle de vie dû à **OSGi**^{TM} permet à des applications, ou services, d'être livrées, mises à jour, installées

[1] Un service contrôlé est un service que l'on vérifie et que l'on modifie régulièrement selon des critères spécifiques.

ou gérées selon le principe du ***just-in-time***. De plus, les services s'enlèvent automatiquement du périphérique lorsqu'ils ne sont plus utilisés, libérant ainsi de la mémoire à d'autres fins.

Dans la version 2.0, JES est conçu pour être conforme avec les spécifications d'**OSGi**^{TM} (version 3.0). Cette version intègre également le produit *Forte for Java* de Sun, et aussi l'infrastructure d'Internet de *iPlanet E-Commerce Solution*, Cette combinaison de technologies permet des solutions ***end-to-end*** pour la livraison de services Internet, nouvelle génération, destinés aux réseaux domestiques.

JES est écrit entièrement en Java et ainsi, il doit fonctionner sur une machine virtuelle Java (version 1.2.2 ou plus). C'est est un ensemble de services de base, tels que *"LogService"*, *"HttpService"*, *"Device"* et *"SSLService"* installés au dessus du noyau JES. On peut développer n'importe quel service dont on a besoin (par exemple, un FaxService, un PrintService etc.) qui peut fonctionner en s'appuyant avec les services élémentaires préexistants de JES. En Regardant le schéma [4.1][GC01], on constate que le JES est la plateforme "HOSTING" sur laquelle des services tournent. Un service est un ensemble de classes et d'interfaces en Java qui permettent d'implémenter un certain dispositif. Prenons, par exemple, le service HTTP élémentaire qui est installé sur le JES, celui-ci constitue la base d'un serveur web minuscule pouvant répondre aux requêtes des clients HTTP. JES possède deux composants principaux. Le premier est un espace de service ServiceSpace^{TM}, qui comprend les API de gestion de vie des services et des applications ***plug-n-play***. Ce modèle modulaire garantit l'installation, le contrôle des numéros de version, la gestion du contenu, la

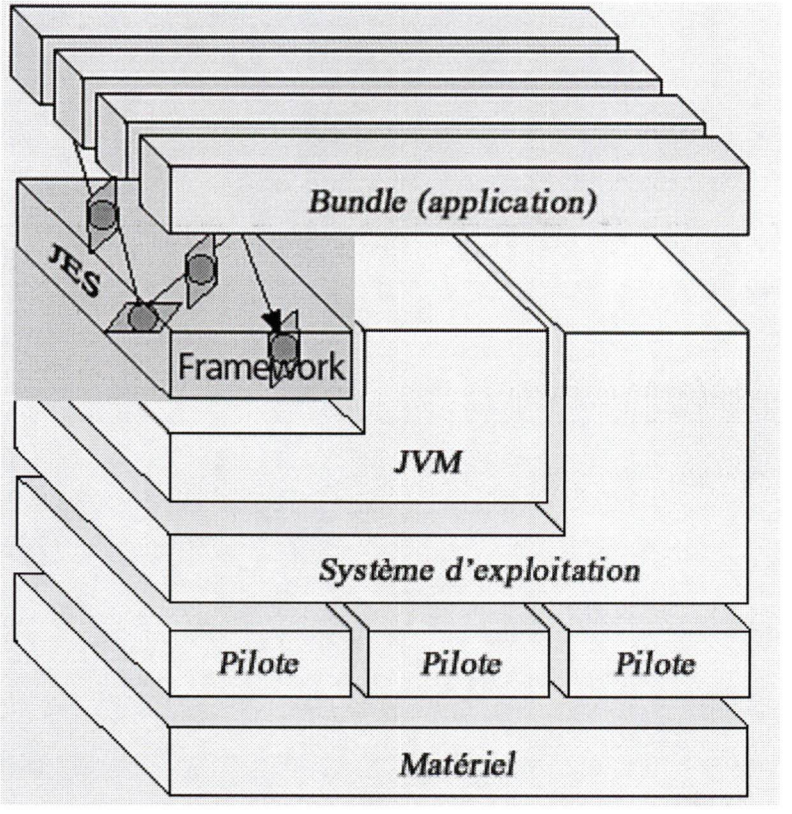


FIG. 4.1 – le Framework *Java Embedded Server*

surveillance, la découverte de services, etc.

Le second groupe de composants est formé, entre autres, des services **OSGi**^{TM} que l'espace ServiceSpace^{TM} invoque et gère. Ces services comprennent HTTP, SNMP, le LOGGING, la gestion des threads, l'administration distante, la prise en charge des servlets, RMI, etc. Les services supplémentaires tels que le courrier électronique ou le fax peuvent être construits au dessus des services de base [SUN00a].

La taille réduite de ce produit, son architecture, son utilisation de la technologie Java et son extensibilité sont principalement destinées aux développeurs et constructeurs d'unités réseau telles que les assistants

personnels numériques (*PDA*), les téléphones cellulaires, les téléphones web, les set-top box, les télévisions, les périphérique médicaux et industriels, les jauges et les compteurs, les distributeurs automatiques de billets de banque, les pompes à essence, les équipements industriels, les équipements et périphériques de télécommunication et équipements de communication réseau tels que les routeurs et commutateurs.

4.3.2 OSCAR

Alors que JES contient quelques services dont le code reste imperceptible par tous les développeurs, OSCAR (*Open Service Container Architecture*) quant à lui, est une véritable implémentation *open-source* de la spécifications d'OSGi^{TM}. Hébergé sur *SourceForge* qui est un service gratuit de *VALinux*[2], son but est de fournir une application conforme et complètement ouverte de la structure d'OSGi^{TM}. à cet effet, OSCAR est actuellement bien conforme avec une grande partie des spécifications d'**OSGi**^{TM} 2.0 dont les fonctionnalités fournies et présentées par OSCAR sont très stables et elles sont utilisées par beaucoup de programmeurs.
Quoiqu'**OSGi**^{TM} vise le marché des unités embarquées, cette structure est idéalement adaptée à l'expérimentation avec le calcul composant-orienté et service-orienté en général. Par exemple, OSCAR peut être facilement injecté dans d'autres projets et être employé comme un *plugin* ou un mécanisme d'extension de services. Il peut assumer cet objectif bien mieux que d'autres systèmes qui sont employés pour des buts semblables, tels que *Java*

[2]VALinux est une société de services américaine spécialisée dans l'environnement Linux.

Management Extentions (JMX)[3].

Actuellement, OSCAR implémente seulement la partie squelette d'OSGi^{TM}. Bien que le but soit de produire des implémentations de service mieux standardisées, OSCAR n'est pas 100% conforme à ce point, à ce niveau-là on peut mentionner les points suivants :

- OSCAR n'est pas encore 100% conforme avec les spécifications d'OSGi^{TM}, mais il a réalisé des progrès considérables depuis sa première implémentation.
- le support actuel des bibliothèques natives n'est pas 100% conforme avec les spécifications d'OSGi^{TM}, qui indiquent que le *Framwork* doit renvoyer une erreur si une bibliothèque native spécifiée n'existe pas sur la plate-forme contenant le paquet concernée. Sur ce point le propriétaire a du réserve puisque ça peut réellement être le cas quand on a besoin d'une bibliothèque native sur une plate-forme, mais pas sur une autre. Les spécifications d'**OSGi**^{TM} apparemment ne permettent pas cette situation mais OSCAR le permet.
- En outre, dans le support de bibliothèques natives, il s'avère que les variables d'environnement ne sont pas toujours cohérentes sur toutes les implémentations JVM. En conséquence, OSCAR peut être difficilement installé sur quelques plateformes.

[3]JMX définit une architecture d'administration des applications Java reprenant les deux niveaux *"manager"* et agent, traditionnels dans le domaine de l'administration. JMX ajoute une couche d'instrumentation de bas niveau faisant le lien avec l'application Java à administrer.

4.3.3 mBedded Server de Prosyst©

Adopté par *ProSyst Software AG*[4], le système mBedded Server [VF02] sous-tendu par plusieurs technologies, entre autres Jini, simplifie la mise en réseau et la connexion à Internet de divers types de terminaux, parmi lesquels les ordinateurs de poche, les imprimantes, les PC, les téléphones mobiles, les téléviseurs, les chaînes stéréo, les réfrigérateurs, les systèmes de chauffage et de climatisation et les systèmes d'alarme. Doté d'une minuscule mémoire de 99 Ko seulement, le *ProSyst mBedded Server* peut être installé dans n'importe quel appareil électronique : un modem, un routeur ou un boîtier pour téléviseur. De là, il peut fournir à tous les autres appareils électroniques appartenant au réseau domestique une foule de services tels que Internet, le courrier électronique, la mise à jour des logiciels et des fonctions d'administration. Le serveur mBedded permet de relier un réseau local à un réseau WAN, et il est capable de manipuler les données (services) indépendamment des normes et des protocoles, cela se réalise en mettant à disposition du programmeur les possibilités suivantes :

- Contrôle et maintenance des unités via l'Internet (la mise à niveau des logiciels).
- Réparation rapide et appropriée des bogues.
- Nouveaux services étendus de contrôle et de sécurité pour les consommateurs et les fabricants.
- Facturation précise des coûts réels des services de gestion.

[4]Fondée en 1997, l'allemande ProSyst Software AG contribue activement par son expertise aux consortiums pour l'établissement des normes standards dans des environnements incluant des technologies, tel qu'*OSGi*^{TM}, *HAVi*, *CABA*, *HomePlug*, forum de Microsoft UPnP etc.

- Nouveaux canaux de distribution (Scénarios innovateurs payer-par-usage).

La solution ProSyst permet aux fabricants, aux entreprises et aux fournisseurs de service, de livrer et de gérer les nouveaux services de contrôle, de mesure, de divertissement, et d'information. En effet, la plateforme mBedded Server fondée sur l'**OSGi**^{TM} offre, outre les services de base d'OSGi^{TM}, une panoplie de services et de technologies, parmi lesquelles on peut mentionner :

* http représentant un serveur web minuscule.
* Bluetooth comme standard libre (sans licence) de communication sans fil entre les équipements électroniques sur une courte distance.
* EHS/Konnex pour *Electronic Home Systems* une première tentative européenne sur le système de bus résidentiel standard, destinée aux installations consommatrices et aux appareils "*Plug-n-play*". Actuellement il fait partie de la norme Konnex.
* DECT *Digital Enhanced Cordless Telecommunications* Norme européenne de transmission radio-numérique pour la téléphonie mobile ou fixe (boucle locale radio).
* Powerline (DPL/PLC) pour le transfert de données et l'accès à Internet en utilisant l'installation électrique classique de l'habitat. À l'aide d'un appareil spécial d'accès (comparable à un modem câble) connecté à la prise de courant et à l'ordinateur, des vitesses d'acheminement allant jusqu'à 3Mbps peuvent être atteintes.
* X10 qui est un protocole de communication permettant l'envoi de commandes via le réseau électrique interne.

L'autre partie de la technologie ProSyst est le mPower Remote Manager qui permet à l'administrateur, d'une part de contrôler facilement les différents appareils, unités et systèmes situés à l'intérieur de la maison, et, d'autre part, de livrer les services à un grand nombre de passerelles/utilisateurs.

CHAPITRE 5

PROTOCOLE UPNP^{TM}

La technologie **UPnP**^{TM} est une architecture distribuée et ouverte de gestion de réseau qui utilise TCP/IP et d'autres protocoles IP pour permettre d'administrer des éléments de proximité, d'exécuter des commandes et de transférer des données sur les dispositifs gérés en réseau dans la maison, le bureau, et les espaces publics.

5.1 DÉFINITION D'UPNP^{TM}

UPnP^{TM} est un ensemble de standards et de technologies pour connecter d'une façon transparente les NAs, les PCs et les services. Cela se fait par l'extension du concept de la connectivité immédiate connu sous le terme *"plug & play"* vers les réseaux et la découverte, la configuration et le contrôle des terminaux de bout en bout. Le forum **UPnP**^{TM} se penche sur l'interopérabilité entre plusieurs appareils et systèmes de types différents. Il se constitue d'un groupe de recherche dont l'intérêt est l'automatisation et la sécurité des habitats, les passerelles Internet, l'imagerie et l'impression, le multimédia et les appareils et périphériques mobiles. L'architecture **UPnP**^{TM}

se base sur des protocoles simples et libres (open source) dont l'objectif est de réaliser l'interopérabilité entre de multiples technologies différentes. Comme exemple, le protocole HTTP permet aux utilisateurs Web de naviguer à travers plusieurs serveurs et plateformes sans qu'ils aient recours à une exécution locale de scripts ou à des changements de configurations. Les fournisseurs de produits supportant **UPnP**^{TM} ne sont pas obligés de se soucier de l'implémentation d'une technologie en particulier, mais plutôt ils sont assurés que leurs produits seront capables d'intéropérer avec un grand nombre de technologies différentes. En outre, la focalisation du groupe **UPnP**^{TM} sur les protocoles filaires libres permet à l'architecture d'être indépendante des langages et des systèmes d'exploitations utilisés. L'avantage de cette approche est qu'elle permet aux développeurs un vaste choix de langages et de plateformes, donc une grande flexibilité pour sélectionner les meilleures infrastructures pour installer leurs produits et une assurance que ces derniers soient intéropérables avec d'autres types.

5.1.1 Architecture UPnP^{TM}

Les composants fondamentaux d'un réseau **UPnP**^{TM} [cf.5.1]sont les dispositifs, les services et les points de contrôle [uF03].

Dispositifs : un dispositif **UPnP**^{TM} est un conteneur de services qui est contrôlé et commandé par des points de contrôle. Un magnétoscope, par exemple, peut être constitué d'un service de *"tuner"* et d'un service d'horloge. Il peut également être composé d'un ensemble de dispositifs embarqués, et dans ce cas le terme "dispositif racine" est utilisé

pour le désigner. Ces dispositifs embarqués peuvent représenter un objet commun ou plusieurs objets de la maison. Du point de vue de l'utilisateur, un dispositif embarqué est traité comme un dispositif racine, avec ses services et ses actions pouvant être exécutées. La particularité se situe au niveau de l'adressage, puisqu'ils auront tous la même adresse IP, celle assignée au dispositif racine.

Services : Les dispositifs encapsulent des services. Ceux-ci sont définis via des actions et des variables d'état. Ainsi, par exemple, un service d'horloge peut avoir une variable current_time qui définit l'état de l'horloge, et deux actions, *set_time* et *get_time* qui permettent de contrôler le service. Un dispositif peut offrir plusieurs services. Chacun d'entre eux est composé de :

- variables d'état
- serveur de contrôle qui reçoit les demandes d'action, les exécute, met à jour les variables d'état et renvoie la réponse si besoin
- serveur d'événements qui informe les entités du réseau **UPnP**^{TM} intéressées d'un changement de l'état du service.

Points de contrôle : Il s'agit de l'entité du réseau **UPnP**^{TM} capable de découvrir et de contrôler les dispositifs de la maison. Après la découverte de dispositifs, un point de contrôle peut :

- récupérer la liste des services associés à un dispositif
- récupérer la description des services d'un dispositif
- invoquer des actions afin de contrôler un certain service
- souscrire aux événements d'un service envoyés au point de contrôle

lors d'un changement de l'état du service.

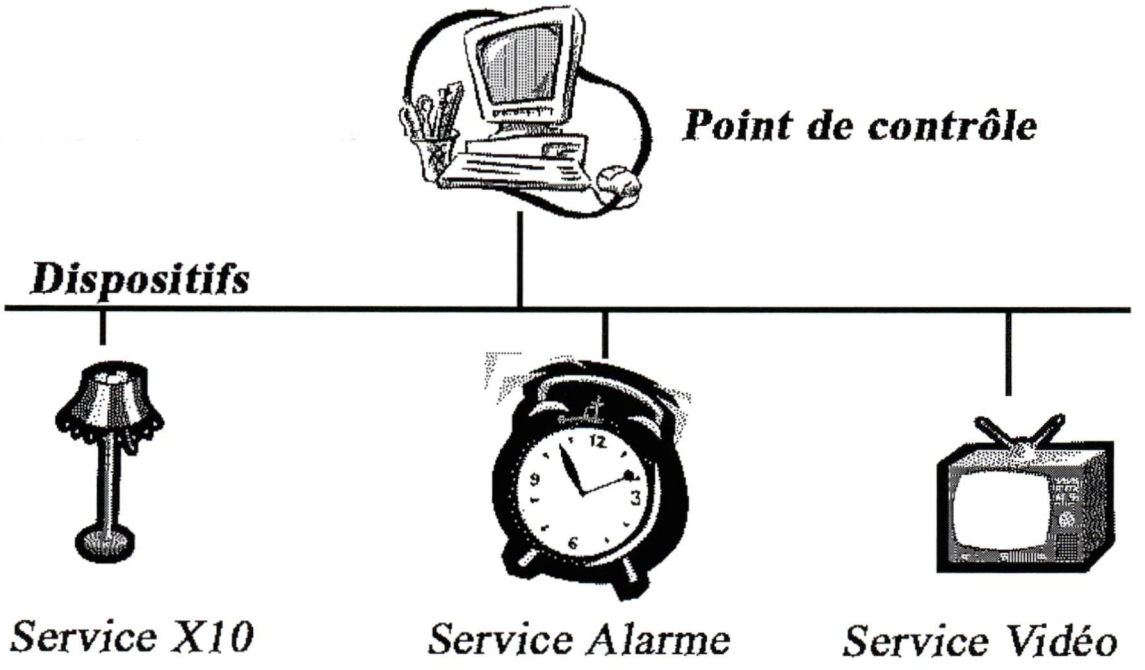


FIG. 5.1 – Composantes d'**UPnP**^{TM}

5.1.2 Fonctionnalités d'UPnP^{TM}

L'architecture **UPnP**^{TM} se compose de cinq services : la découverte, la description, le contrôle, l'annonce et la présentation.

Découverte (*discovery*) : c'est un protocole permettant à un appareil connecté sur le réseau d'avertir les services de contrôle qui y sont liés. Réciproquement, quand un point de contrôle est ajouté au réseau le service de découverte lui permet de trouver les dispositifs qui l'intéressent. Le message de découverte contient les informations essentielles sur le dispositif notamment son type, son identificateur

réseau et un pointeur vers d'autres informations plus détaillées, en plus des services qu'il fournit. Ce service se base sur le protocole SSDP pour *Simple Service Discovery Protocol.*

Description (*description*) : quand un point de contrôle découvre l'existence d'un nouveau dispositif, il a besoin de connaître plus de détails sur celui-ci. Il se sert alors de l'URL fourni lors de la découverte. Cet URL pointe sur un document XML de description contenant l'identification du produit, à savoir, le nom du modèle, le numéro de série, le manufacturier etc. Les dispositifs peuvent contenir des périphériques logiques embarqués et des services. Une liste de ces composants et celle des variables d'état du dispositif sont également incluses dans le document de description **UPnP**^{TM} correspondant. Pour chaque service, la description contient une liste des actions, et pour chaque action, l'ensemble de ses arguments.

Contrôle (*control*) : le point de contrôle peut demander l'exécution d'une action quelconque dans un dispositif après qu'il ait trouvé sa description. Pour cela, il envoie un message de contrôle à l'URL de contrôle fourni dans la description relative au service demandé. Les messages de contrôle sont exprimés en XML et ils se basent sur SOAP (Simple Object Access Protocol) [schéma.5.2]. Le périphérique renvoie les résultats des actions demandées et ses variables d'état changent de valeurs.

Notification (*eventing*) : quand les variables d'état d'un dispositif changent, celui-ci informe des nouvelles valeurs par l'envoi d'un

message d'événement encodé en XML via HTTP. Ce message contient la liste des variables d'état du dispositif et leurs valeurs instantanées. Ainsi, chaque nouveau point de contrôle inscrit peut recevoir cette mise à jour. Cette inscription est établie en se basant sur l'architecture GENA (General Event Notification Architecture). Il existe un message d'événement spécial, destiné souvent au point de contrôle qui s'inscrit pour la première fois. Il contient la liste de toutes les variables d'état pouvant connaître des changements. Il permet au point de contrôle en question d'initialiser son modèle d'état du service correspondant. Afin de supporter un système à plusieurs points de contrôle, le service d'événements cherche à garder tout le système équitablement informé de l'état actuel du dispositif en permanence.

Présentation (*presentation*) : ce service basé sur HTML et intégré dans un dispositif **UPnP**^{TM} permet à un utilisateur de le contrôler depuis son navigateur standard. Un point de contrôle détermine l'URL de la présentation du dispositif depuis son document descriptif. Ensuite il charge la page web pointée par cet URL sur le navigateur. Puis, il commence la gestion utilisateur du périphérique. La couche application constitue un composant additionnel du réseau **UPnP**^{TM}. Il s'agit d'un module logiciel spécifique[1] permettant de connecter un dispositif au réseau UPnP et, par la suite, de le contrôler à distance.

Les modèles de description de dispositifs peuvent être préparés soit par leur propre fabricant lui-même, soit par d'autres fabricants participant

[1]Les options offertes par le module logiciel dépendent des capacités que le dispositif contrôlé possède, ainsi que des règles spécifiées selon les objectifs du programmeur.

au forum de **UPnP**^{TM} afin de produire des standards industriels et des modèles de services. Le comité **UPnP**^{TM} actuel a déjà préparé des définitions de modèles standardisées. Il comporte les comités de recherche en Audio-visuel, en passerelle Internet, et en imagerie et impression.

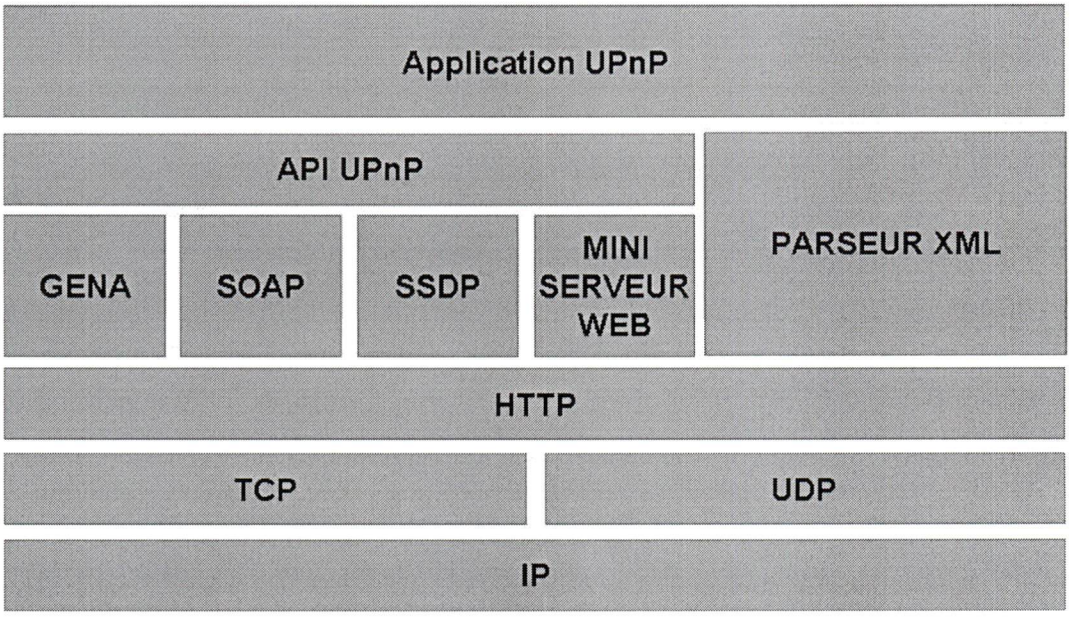


FIG. 5.2 – Architecture de la pile **UPnP**^{TM}

CHAPITRE 6

PROTOCOLE X10

6.1 PRINCIPE

La principale caractéristique du protocole X10 est qu'il permet de transférer des données via la porteuse standard du courant alternatif, en transmettant des segments d'une milliseconde de longueur chacun, ensuite les segments superposent le signal d'information à celui du courant [Ide03]. L'utilisation de la technologie X10 permet de bâtir un environnement intelligent contenant des équipements pouvant être facilement contrôlés depuis un point central distant [cf.Fig.6.1]. Physiquement, l'interaction entre un PC et un terminal X10 sur le réseau, est fournie par l'interface *SERIAL-TO-X10* qui permet de traduire les données envoyées par le PC à un format compatible X10, et de les envoyer au module correspondant par la suite. Cette interface est attachée d'un coté, au port série du PC, et de l'autre coté, à la ligne du courant électrique. En général, une application qui permet de recevoir, traiter, et envoyer des requêtes clientes depuis le PC vers l'interface SERIAL-TO-X10 peut être considérée comme une passerelle X10.

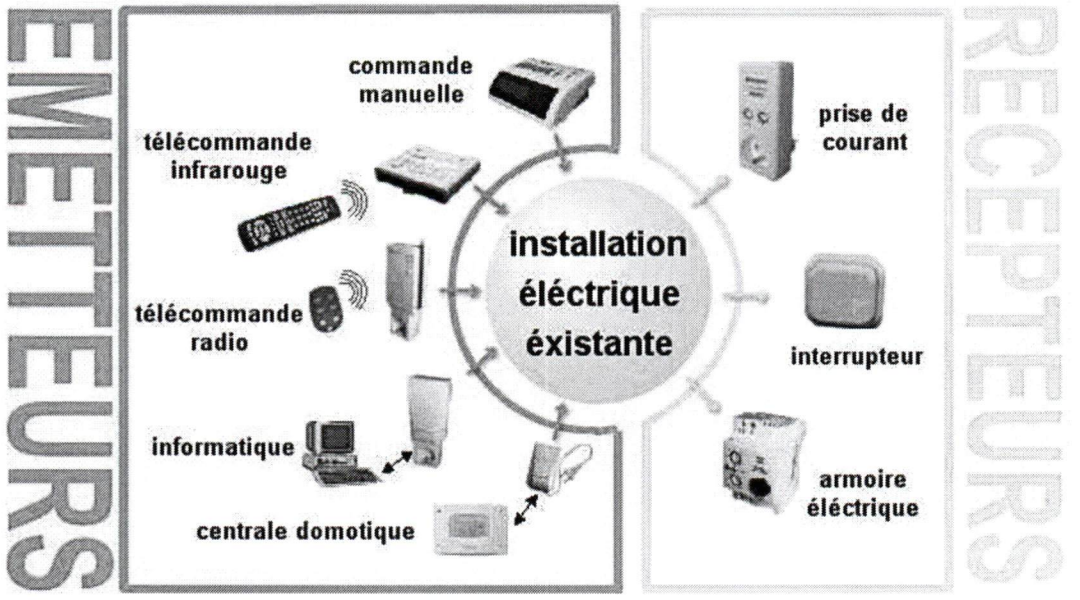


FIG. 6.1 – Réseau X10 dans un habitat

6.2 MODULES X10

Pour être capable d'exploiter les capacités d'un réseau X10 , les équipements doivent être branchés au courant électrique du bâtiment. Chacun d'entre eux possède une adresse, et par la suite, il est possible d'ordonner à l'appareil d'accomplir une tâche précise. Le protocole X10 permet de gérer 256 modules (par contrôleur)[Kin99]. Chaque module possède un identificateur maison "*house code*" réglable de A à P, et un identificateur unité "*unit code*" réglable de 1 à 16, ce qui fait 16x16=256 possibilités [cf.Fig.6.2]. Mais la capacité d'adresser un certain nombre donné de modules est liée à l'émetteur :

* 16 modules (1 code maison) avec une télécommande "*8 in 1*"
* 256 modules avec une interface CM11 connectée à un PC

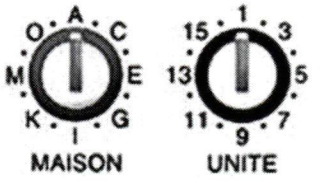


FIG. 6.2 – Adressage X10

Il existe trois types de modules X10 :

1 Modules émetteurs :

* Interface PC (CM11) module émetteur / récepteur permettant de contrôler tous les modules récepteur d'une habitation à partir d'un PC. C'est donc le contrôleur X10 "*CM11A*" qui gère tous les appareils X10 en maintenant une table interne qui garde une trace de leur état. Cette table est mise à jour dès que leur état change.
* La télécommande radio : cette télécommande envoie des commandes radio (RF) à un module qui convertit les commandes reçues au courant porteur (RF $\Rightarrow$ CP) permettant ainsi de contrôler des modules récepteur de l'habitation à partir d'une télécommande

2 Modules récepteurs :

* Le module appareil (AM12) permet une fonction MARCHE / ARRÊT
* Le module lampe (LM12) permet les fonctions MARCHE / ARRÊT et VARIATION de l'intensité lumineuse

3 Modules détecteurs et convertisseurs :

Il existe de nombreux modules permettant la détection : contact de porte, bris de glace, mouvements, lumière. D'autres dont le rôle est la conversion des ordres X10 : radio vers courant porteur, infrarouge vers courant porteur.

Selon les modules utilisés, les commandes suivantes peuvent être envoyées :

* Marche (on)
* Arrêt (off)
* Baisser (dim)
* Augmenter (brighten)
* +5, +10, ...+100
* -5, -10, ...-100
* ON/OFF (tous les modules)

Le protocole $\widehat{\text{X10}}$ est un protocole qui se base sur la configuration et les commandes utilisateur. Cela dit que le protocole attend toujours que l'utilisateur lui indique l'existence et l'identificateur d'un appareil X10 quelconque dans l'habitation. Cela est dû au fait que $\widehat{\text{X10}}$ ne comprend pas un service de découverte lui permettant d'identifier automatiquement les appareils X10 aussitôt que ceux-ci sont branchés sur le réseau.

Dans la prochaine partie nous verrons comment les différentes technologies, décrites plus haut, ont été mises en œuvre dans une application de test.

Troisième partie

Mise en œuvre

CHAPITRE 7

INTRODUCTION

L'implémentation de la démonstration du présent sujet a été entièrement développée en Java, et est passée par plusieurs étapes. En effet, comme il s'agit de l'intégration d'une multitude de technologies, il fallait étudier et mettre en œuvre chaque technologie séparément, afin de se familiariser avec elle, avant de l'intégrer dans l'application "mère" de ce projet. Les technologies en question sont : la plateforme **OSGi**^{TM} [chap.4], le protocole X10 [chap.6], le protocole **UPnP**^{TM} [chap.5] et le protocole SIP [chap.3].

La première étape consistait à tester la plateforme **OSGi**^{TM} sur un ordinateur. Le produit qui a été élu pour cette tâche est JES [sect.4.3.1]. La version 2.0 a été téléchargée depuis le site de SUN© et a été mise en œuvre sur la machine virtuelle Java (jdk 1.4). Le produit de SUN© met en disponibilité un certain nombre d'exemples et de services de base sous formes de *bundles*, afin de bien comprendre le bon fonctionnement de la plateforme JES, et de mettre en disponibilité des ressources de base pour d'éventuelles applications plus avancées. Les principaux services de base sont :

- Service HTTP : pour les applications liées au développement Web, surtout

celle qui requiert l'interrogation d'un serveur Web comme les servlets

- Service Log : pour l'identification et l'authentification.

Dans la deuxième étape, le du protocole $\widehat{\text{X10}}$, a joué le rôle d'un réseau domotique interne d'un habitat. Les outils de programmation en Java disponibles pour implémenter une application utilisant ce protocole utilisent en général le port série, puisque la majorité des contrôleurs X10 se connectent à l'ordinateur via son port série. Le paquetage "*Javacomm*"[1] (version 3.0) de SUN© comporte les outils nécessaires pour communiquer et contrôler le port série, et par la suite, tout appareil connecté à ce port.

Pour la troisième étape, la question qui se posait était : "*comment gérer un environnement résidentiel comportant plusieurs technologies pour contrôler différents types d'appareils domestiques, dont* $\widehat{\text{X10}}$ *?*". Pour répondre à cette question, nous avons fait un survol sur les différentes technologies de contrôle résidentiel, et ensuite nous avons choisi celle qui répond le mieux aux deux critères suivants :

1 Agir comme nœud central reliant plusieurs technologies hétérogènes cohabitant dans le même domaine

2 Servir de point d'accès à l'ensemble de ces technologies depuis l'extérieur.

La technologie qui a été retenue pour répondre aux exigences mentionnées est **UPnP**^{TM} [chap.5].

La communication distante avec l'habitat est une tâche qui exige de prendre en compte un certain nombre de facteurs tels que la sécurité,

[1]http ://java.sun.com/products/javacomm/

l'interopérabilité etc. Le protocole SIP a été utilisé comme outil de communication et d'échange de messages avec les appareils domestiques distants. L'intégration de cet outil dans cette application était le sujet de la quatrième étape, à la fin de laquelle toutes les autres technologies ont été intégrées pour réaliser la démonstration liée à ce sujet.

CHAPITRE 8

DÉPLOIEMENT DE LA PLATEFORME **OSGi**^{TM} SUR PC

Comme il a été déjà mentionné auparavant, le produit à base d'OSGi choisi est JES, à cause de sa flexibilité surtout, mais aussi de son appartenance au propriétaire du JDK. Cela lui permet d'être commode avec l'interprétation du code Java. Ainsi, un programmeur, familiarisé avec Java, peut facilement comprendre les messages d'erreurs affichés sur sa console JES, et qui l'empêchent d'installer un *bundle* puisqu'il s'agit souvent du même genre d'erreurs qu'il peut affronter lors d'une compilation usuelle avec un *Javac*, s'ajoutent à ce genre quelques erreurs liées à la spécification de la plateforme **OSGi**^{TM} elle-même, comme les erreurs liées à l'écriture des entêtes du fichier *Manifest*[1] [cf.Tab.8.1] ou bien lors d'un échec d'exécution d'une requête lue dans le fichier *Manifest*, comme, par exemple, l'importation/l'exportation d'un service ou d'un paquetage [oA03].

[1] *Manifest* est le fichier descripteur du *bundle*, il contient des en-têtes permettant de décrire et de gérer le comportement du *bundle* quand celui-ci est installé.

8.1 INSTALLATION ET LANCEMENT DE LA PLATEFORME JES

Avant de procéder à une installation de **JES**, il est indispensable d'avoir la machine virtuelle JAVA^{TM}préinstallée sur le système d'exploitation. Le dossier d'installation de JES contient les dossiers suivant : *bin, docs, lib,* et *bundles*. Les interfaces et les classes de base de la plateforme **OSGi**^{TM}, ainsi que leurs implémentations sont localisées dans le fichier */lib/framework.jar*. Ce fichier, dont le chemin doit être ajouté à la variable *CLASSPATH*, est nécessaire pour construire les différents *bundles* et pour faire fonctionner **JES**. Le dossier */bundles* quant à lui contient les *bundles* livrés avec **JES**, tels que *Log* et *HTTP*. Pour lancer **JES**, il suffit d'exécuter le batch *"runjes"* fourni avec le système de bas de JES, et qui permet de redéfinir la *CLASSPATH* et d'exécuter la commande : *java com.sun.jes.impl.framework.Main*[2].

8.2 CRÉATION D'UN *bundle*

En général, chaque *bundle* a besoin au moins d'un fichier *Manifest* et d'une classe activatrice [GC01][SUN00b]. Ainsi, le lancement d'un *bundle* doit passer par les étapes suivantes :

Étape 1 : Écriture de la classe activatrice : qui doit implémenter l'interface "*org.osgi.framework.BundleActivator*" et redéfinir les deux

[2]Il s'agit ici de l'environnement Windows. Pour Unix il faut suivre les consignes de configuration décrites dans le tutoriel JES de SUN© .

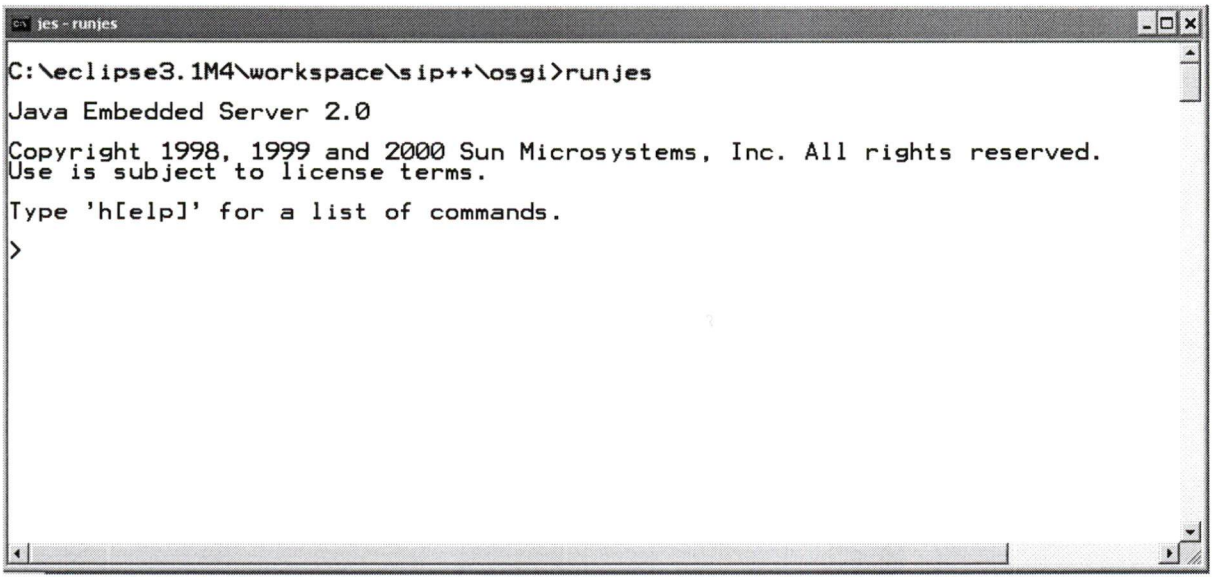


FIG. 8.1 – Lancement de Java Embedded Server 2.0

méthodes : *"start" et "stop"* que **JES** appelle pour lancer et arrêter un *bundle*

Étape 2 : Création du fichier *Manifest* : tout *bundle* installé sur la plateforme JES doit contenir un fichier texte nommé *Manifest* décrivant son contenu. Il est constitué de plusieurs entêtes indiquant à la plateforme les liens vers les fichiers et les ressources que le *bundle* encapsule. Le tableau [Tab.8.1] indique quelques exemples d'entêtes que peut contenir un tel fichier

Étape 3 : Création du fichier JAR du *bundle* : contenant le fichier *Manifest* et la classe activatrice[3]

Étape 4 : Installation et lancement du *bundle* : sur la console [Fig.8.1] où **JES** a été lancé, l'installation du *bundle* "*bundle1.jar*" s'effectue à

[3]Ici c'est le cas le plus simple. Éventuellement le fichier JAR peut contenir d'autres ressources tels que d'autres fichiers JAR par exemples.

partir de la commande suivante :

```
>install dossier_contenant_le_bundle1/bundle1.jar
```

Il ne reste qu'à récupérer l'identificateur '*ID*" du *bundle* et de le lancer grâce aux lignes de commandes suivantes :

```
>bundles
  ID  STATE       LOCATION
  --  ---------   -------------------------
  1   INSTALLED   dossier_contenant_le_bundle1/bundle1.jar
  2   INSTALLED   dossier_contenant_le_bundle2/bundle2.jar
>start ID (ici c'est '1')
```

Entête	**Signification**
Import-Package	Packages requis
Export-Package	Packages fournis
Import-Service	Services requis
Export-Service	Services fournis
Bundle-Activator	Nom de la classe Activator
Bundle-ClassPath	Emplacement des classes et des ressources du bundle
Bundle-NativeCode	Bibliothèque natives à charger en fonction du processeur, du SE
Bundle-UpdateLocation	URL de mises à jour du bundle
Bundle-Name	Nom du Bundle
Bundle-Description	Description du Bundle
Bundle-Version	Version du bundle
Bundle-DocURL	URL de la documentation du bundle
Bundle-ContactAddress	Coordonnée du propriétaire du bundle
Bundle-Copyright	Copyright du bundle
Bundle-Category	Catégorie du bundle

TAB. 8.1 – Les entêtes possibles du fichier Manifest

8.3 Cycle de vie d'un *bundle*

Pour chaque *bundle*, un "*Bundle Object*" est associé. Il permet de gérer le cycle de vie du *bundle*. Il contient également un certain nombre d'informations telles que :

- Identifiant du bundle
- Localisation du bundle
- État du bundle qui est variable et qui peut prendre l'une des valeurs suivantes :
  - ***INSTALLED*** **:** le *bundle* est correctement installé
  - ***RESOLVED*** **:** le *bundle* est en attente, mais il peut démarrer car tous les services nécessaires sont disponibles
  - ***STARTING*** **:** le *bundle* est en train de démarrer
  - ***STOPPING*** **:** le *bundle* est en train de s'arrêter
  - ***ACTIVE*** **:** le *bundle* est actif, il est en cours de fonctionnement et est entrain d'exécuter les tâches demandées
  - ***UNINSTALLED*** **:** le *bundle* est désinstallé.

Le schéma [Fig.8.2] décrit le cycle de vie d'un *bundle* avec les transitions possibles d'un état à un autre.

Les prochains chapitres décrivent, en détails, les différents *bundles* développés et installés au cours de ce projet, les divers services dont ils ont besoin et les services et les paquetages qu'ils offrent.

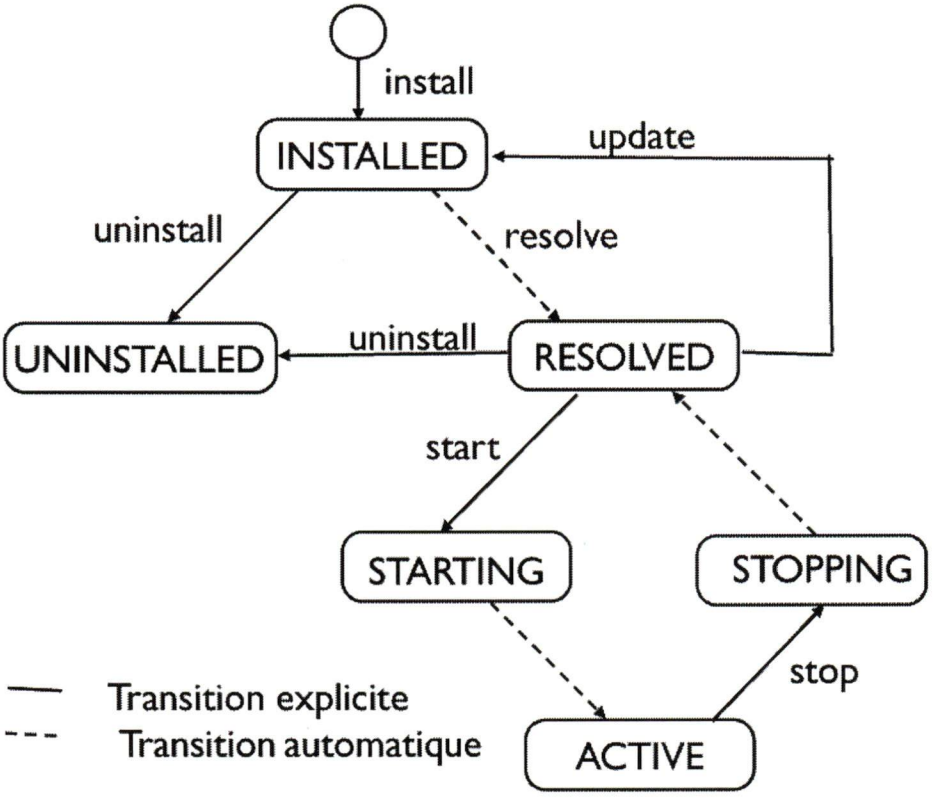


FIG. 8.2 – Cycle de vie d'un *bundle*

CHAPITRE 9

INTÉGRATION DU CONTRÔLE X10 SUR UN PC

9.1 INTRODUCTION

Dans ce chapitre nous mentionnons comment communiquer avec un contrôleur X10 via le port Série (COMX). Une grande partie de cette application a été inspiré du site "le projet Java X10 *(The Java X10 Project)*" [Was03], qui requiert l'installation du module *JAVACOMM*. Le site offre une *API* Java (*Open Source*) qui sert d'une base réutilisable et extensible pour toute application visant à communiquer avec un module ou un contrôleur X10. Plusieurs modifications ont été effectuées sur cet *API* afin de l'exécuter sur la plateforme OSGi^{TM}.

En général, toute application Java d'automatisation avec X10, qui intègre le protocole $\widehat{\text{X10}}$ doit être composée de trois principaux paquetages :

1. **Application X10 :** c'est l'application d'automatisation en tant que telle. Éventuellement, elle pourrait être constituée de plusieurs paquetages et éléments logiciels, entre autres, l'interface utilisateur et d'autres

technologies de réseautage. Une description détaillée de cette partie sera présentée dans les prochaines sections

2. Java X10 : il s'agit des paquetages du projet Java X10 [cf.section9.2]

3. Java comm : c'est un ensemble de modules et d'*API* Java développés par SUN© pour accéder aux ports (COMX) [cf.section9.3].

Le schéma UML suivant [Fig.9.1] indique une vue générale des trois éléments logiciels indiqués plus haut :

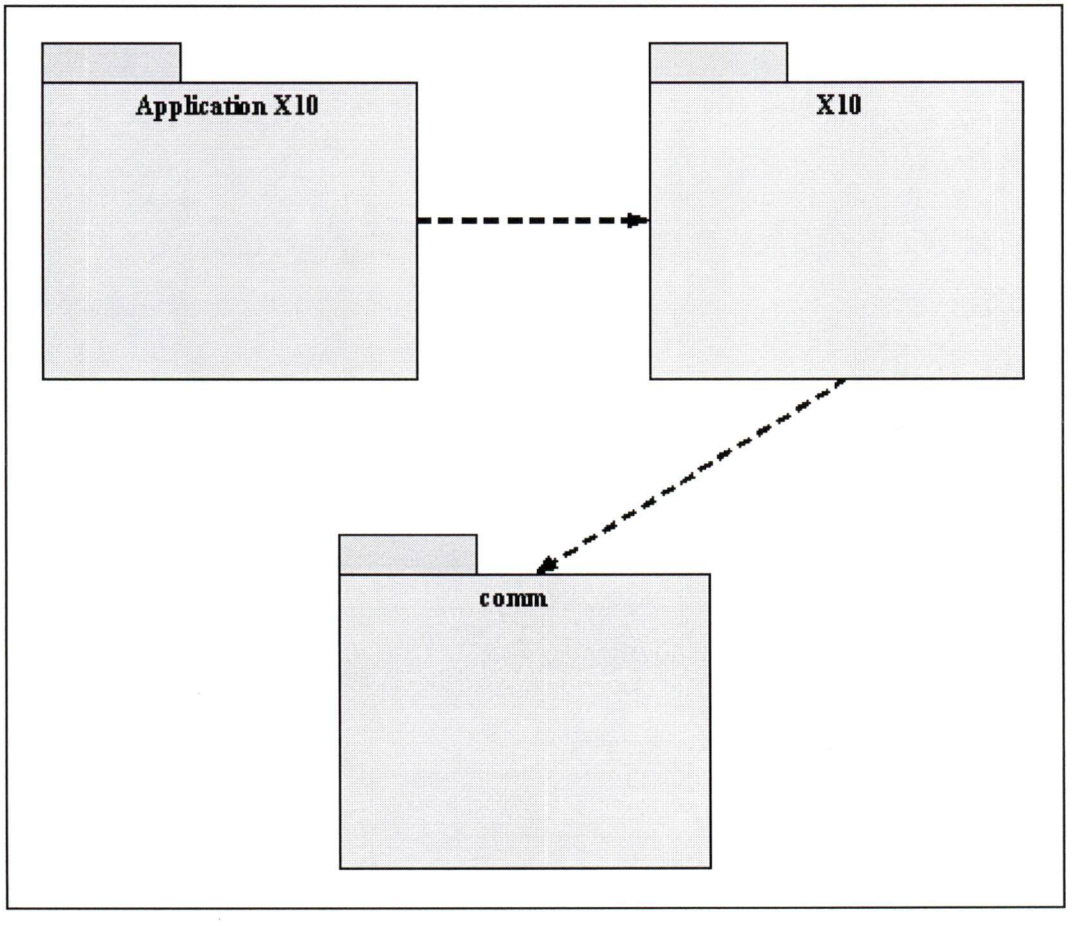


FIG. 9.1 – Paquetages nécessaires pour toute application X10

9.2 PROJET JAVA X10

Le projet Java X10 [Was03] est un ensemble d'*APIs* (*Open Source*) fournissant la base nécessaire à une application désirant communiquer avec un contrôleur X10 (*CM11A*). Cet *API* comporte essentiellement les classes qui constituent le coté client d'une application X10 communicant avec le serveur, qui est le contrôleur X10. Cette communication, entre le client et le contrôleur, doit être basée sur les protocoles TCP/IP et elle doit utiliser les *sockets*. Ainsi, les *APIs* Java X10 permettent à toute application cliente X10 écrite en Java, d'interagir avec les appareils X10.

Les *APIs* X10 sont compactés dans le fichier source "*X10.jar*" que l'on doit installer sur le système comme élément de la librairie des sources de l'application. Son rôle est de recevoir toutes les commandes issues de l'interface utilisateur (premier paquetage), de les traduire adéquatement en actions et événements X10 et de les acheminer vers le paquetage "*comm*". Réciproquement, le paquetage X10 permet également d'accueillir les informations requises (événement, état etc) depuis un module X10 et de les rediriger vers l'application cliente. Le schéma [Fig.9.2] montre comment une application X10 gère un appareil domestique (X10) en communicant avec le contrôleur *CM11*.

9.3 INSTALLATION DU MODULE *JAVACOMM*

Le module *JAVACOMM* est un produit de SUN© composé de plusieurs éléments logiciels "multi-niveaux" distribués en trois couches de la façon

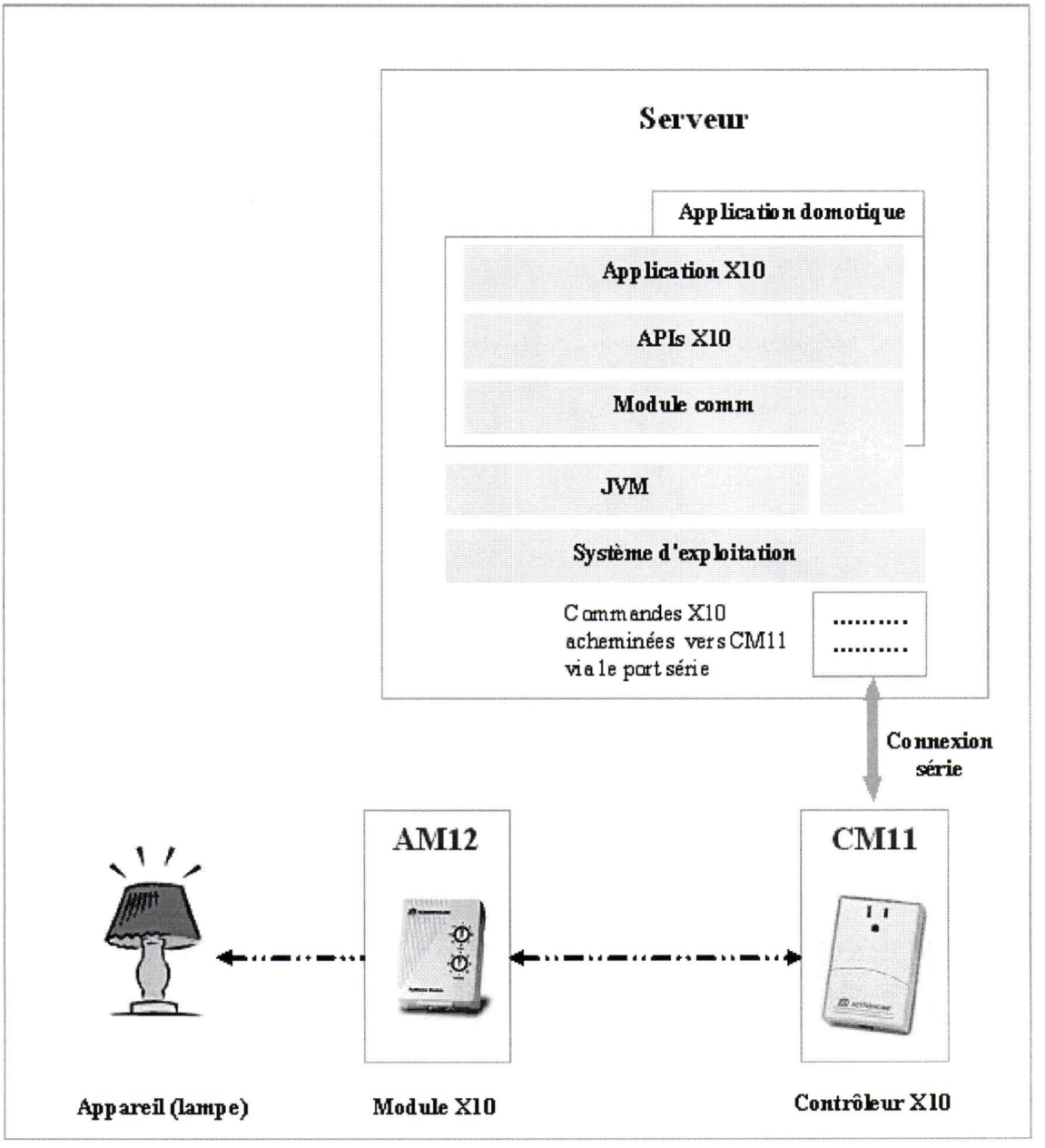


FIG. 9.2 – Interaction entre une application X10 et un contrôleur CM11

suivante :

- les classes ***haut-niveau*** qui sont des applications pour gérer et tester les ports séries et parallèles

- les classes ***bas-niveau*** qui sont des interfaces entre la machine virtuelle et la partie physique du système
- les classes ***pilotes*** (*Drivers*) qui constituent une couche entre les classes bas-niveau et la partie la plus basse du système[1]. Bien que cette partie soit une composante essentielle du produit, son code source associé, par contre, n'est pas accessible par le développeur.

Dans le manuel d'utilisateur de ce module[2], et afin de pouvoir accéder aux ports de communication via une application s'exécutant directement sur la machine virtuelle Java, il est recommandé d'appliquer strictement une méthode spécifique[3].

Outre les deux dossiers : "*javadocs*" pour la documentation sur les classes Java et "*samples*" qui comporte des exemples élémentaires pour tester les ports, le dossier "*commapi*" contient également trois fichiers essentiels pour le fonctionnement de toute application sur les ports (COMX) :

1 ***comm.jar*** : qui contient les deux paquetages "com.sun.comm" et "javax.comm" (classes *bas-niveau*). Ce dernier doit être importé par toute classe servant à manipuler les ports (COMX) et il doit être copié dans le dossier <JDK>lib

2 ***win32com.dll*** : est la bibliothèque qui fournit un accès direct aux pilotes système des ports (COMX). Ce fichier doit être installé sur le dossier *<JDK>/bin*

[1]À titre d'essai, ces modules ont été exécutés sur le système Linux avec succès après l'installation du driver RXTX, associé à Linux, qui permet de communiquer avec le port série sur cette plateforme

[2]fourni dans la documentation de SUN©

[3]Les étapes de cette méthode sont citées en détails dans le manuel. Cependant, si l'application s'exécute sur la plateforme OSGi^{TM}, il faut procéder suivant une autre méthode décrite plus loin dans ce document.

3 ***javax.comm.properties*** : un fichier texte contenant une seule ligne : "*Driver=com.sun.comm.Win32Driver*". Il doit être copié dans le dossier *<JDK>/lib*.

9.4 INSTALLATION DU PROTOCOLE X10 SUR LA PLATEFORME OSGi^{TM}

Dans cette section nous allons expliquer ce qui a été développé pour l'intégration du protocole X10 sur la plateforme OSGi^{TM}. Plusieurs exemples du *Projet Java X10* ont été testés pour en choisir un. L'élément le plus important qui a été retenu de ces exemples, outre le paquetage X10, est la classe "*X10SerialServer*" qui, une fois installée sur la station connectée au contrôleur X10, joue le rôle du serveur X10. Toute application désirant accéder à ce contrôleur doit se connecter à ce serveur en utilisant les *Sockets* (application *Client-Serveur*). L'installation du service X10 sur la passerelle **OSGi**^{TM} dépend essentiellement du serveur X10. Dans ce qui suit nous décrivons principalement la création du *bundle X10Server*.
Comme tout *bundle* construit dans ce projet, la création du *bundle X10Server* a suivi les mêmes étapes décrites dans le chapitre 8 [cf.sect.8.2].

Écriture de la classe activatrice : Il s'agit d'écrire la classe "*Activator*" qui implémente l'interface "*BundleActivator*". Comme il a été déjà mentionné, cette classe intermédiaire entre le *bundle* et la plateforme, permet de

transmettre à celle-ci les actions que l'objet implémenté veut lancer [GC01]. Dans le cas du *bundle X10Server*, il s'agit de créer une instance du serveur X10, et de le mettre à l'écoute de toute communication cliente demandant l'accès au contrôleur X10 [cf.Annexe.A.1]. En suite, un service nommé *X10ServerService* est créé et est fourni par ce *bundle*. La création d'un service, en génétral, commence par l'écriture d'une classe où l'on spécifie ce que ce service est en mesure de faire comme tâches. Dans notre cas, le service *X10ServerService* aura la charge d'exécuter les commandes des clients sur le contrôleur X10. Chaque commande communiquée au serveur est, après son interprétation, envoyée au contrôleur qui l'achemine à l'appareil approprié. L'exécution des tâches demandées au *bundle* ainsi que l'enregistrement du service créé se fait au cœur de la méthode "*start*" de la classe activatrice qui, dans la majorité des cas, joue le même rôle que celui de la méthode "*main*" au cas où l'application est lancée sur la *JVM* directement (la commande *java*). La principale différence entre les deux méthodes, est que "*start*" n'attend jamais d'arguments de la part de l'utilisateur, contrairement au cas du "*main*", où l'utilisateur peut spécifier les arguments qu'il veut que son application prenne en compte dans son exécution. Pour remédier à ce problème, différentes façons de spécifier des arguments à un *bundle* sont possibles. À titre d'exemple, l'utilisateur peut éditer tous les arguments désirés dans un fichier texte. Le lien vers ce fichier est indiqué au *bundle* d'une manière qui, pratiquement, dépend de son emplacement. Ainsi :

- Si le ficher est encapsulé dans le *bundle* lui-même, le changement de son emplacement est quasiment rare. Donc le lien vers celui-ci peut être fixé

dans le code de la classe java.

- Si ce fichier est stocké dans un emplacement, autre que le *bundle*, le lien vers celui-ci est indiqué, soit directement, en spécifiant le chemain dans la classe activatrice, soit en passant par les propriétés. Dans cette dernière méthode on indique à la classe activatrice le nom de la propriété contenant le chemin. Pour cela, un objet de type *Properties* doit être créé, ensuite, la valeur de la proporiété est ajoutée ou récupérée en utilisant les deux méthodes *put* et *get*. Sur la console de JES cette propriété peut être éditée, grâce à la ligne de commande :

  ```
  >add <propriété> <valeur>
  ```

  La méthode des propriétés est plus flexible car elle permet de changer la valeur de la propriété directement sur la console sans changement de code. Cela permet d'éviter de recommencer plusieurs fois la reconstitution du *bundle* [4].

Création du fichier *Manifest* : L'écriture du contenu du fichier *Manifest* constitue une étape cruciale dans le processus de la création d'un *bundle*. En effet, ce fichier contient toutes les informations nécessaires qui décrivent les ressources que le *bundle* offre ou dont il a besoin. Dans le cas du *bundle X10Server*, en dehors des paquetages à importer des autres *bundles*, les ressources qui sont nécessaires pour installer ce serveur sont les composantes du *JAVACOMM*, en particulier, les deux fichiers : commwin32.jar et win32com.dll [cf. Annexe A.2].

Pour le cas de *X10Server* le schéma [cf.A.2] dans l'Annexe montre le contenu

[4]Chaque fois que les classes appartenant à un *bundle* soient changées, il faut une recompilation de ces classes et par la suite une reconstruction de ce *bundle*.

du fichier *Manifest* associé. Dans ce fichier, outre l'entête qui indique le nom et le chemin de la classe activatrice (*Bundle-Activator*), les entêtes primordiales pour l'installation de ce *bundle* sur la passerelle OSGi^{TM}, sont *Bundle-ClassPath* et *Bundle-NativeCode*. La premiere sert à signaler au *framework* la nouvelle valeur *CLASSPATH* que le *bundle* doit prendre comme référence pour l'exécution de ses classes. Il s'agit principalement des deux fichiers JAR :

1. ***commwin32.jar*** : qui n'est autre que l'*API Javacomm*. Ici, il est essentiel d'inclure cette archive dans le *bundle* lui-même et de le publier dans le *Manifest*, sans quoi, il ne pourrait pas le localiser, et par la suite, l'installation échouerait.

2. ***upnp-x10.jar*** : est un fichier contenant, à la fois, les paquetages relatifs au projet X10 [cf.9.2], et ceux du protocole **UPnP**^{TM} que l'on décrira dans chapitre 10. Nous avons choisi d'unir les deux paquetages ensemble pour faciliter la communication entre les deux protocoles et pour gagner du temps durant l'exécution de l'application[5].

En ce qui concerne l'entête *Bundle-NativeCode*, là encore, il s'agit d'une autre composante du module *Javacomm* qu'il faut inclure et référencer dans le *bundle* , à savoir le fichier win32com.dll, en plus des autres informations comme le système d'exploitation et l'architecture du processeur utilisés. Ces informations sont indiquées dans cet entête en particulier, car il décrivent une partie basse du système et du code natif que le *bundle* utilise pour s'exécuter. En général, cet entête n'est spécifié que s'il y a du code natif que

[5]L'exécution manuelle de chaque bundle sur la console ajoute un temps suplémetaire variable.

le *bundle* utilise pour accéder à la partie basse du système. Ce code natif pourrait être des fichiers portant les extensions : .dll, .c etc.

CHAPITRE 10

DÉPLOIEMENT DU PROTOCOLE UPNP^{TM} SUR LA PLATEFORME OSGI^{TM}

10.1 INTRODUCTION

L'**UPnP**^{TM} (*Universal Plug and Play*) est un ensemble de protocoles destinés aux réseaux des dispositifs (*devices*). L'objectif de cette étape était de parvenir à intégrer le protocole **UPnP**^{TM} dans une passerelle résidentielle afin de contrôler les dispositifs domestiques internes. Différents outils de programmation sont fournis afin de faciliter le développement, l'extension et l'intégration du protocole **UPnP**^{TM}. Parmi les plus importants de ces outils, si ce n'est pas le plus important de tous, est le paquetage Java de *Cyberlink*. C'est ce dernier qui a été choisi pour être intégré dans ce projet, non seulement pour la clarté de la documentation qui fournie, mais aussi principalement, pour sa structure qui est une structure orientée service, ce

qui est bien commode si l'on désire utiliser la spécification de la plateforme **OSGi**^{TM}.

10.2 PAQUETAGE JAVA UPNP DE *Cyberlink*

Le paquetage Java de *Cyberlink* appelé (*CyberLink for Java*) est un ensemble d'*APIs* mis à la disposition des développeurs **UPnP**^{TM}. Le développeur de *Cyberlink* [Sat02] assure un contrôle et un support pour ces *APIs* régulièrement afin d'aider les programmeurs à créer différents types de dispositifs et de points de contrôle. Le paquetage comporte également plusieurs exemples d'application pour ce protocole. Il est constitué essentiellement de trois types d'unités réseautiques : les dispositifs (*devices*), les services et les points de contrôle [cf.5.1.1]. Une description technique détaillée de ces trois composantes fera l'objet des prochaines sections. Il faut noter d'abord que l'installation et l'utilisation de ce paquetage requiert l'utilisation d'un parseur *XML*, puisque les requêtes d'**UPnP**^{TM} et surtout de *SOAP* sont en *XML*.

10.2.1 Programmation des dispositifs

En **UPnP**^{TM}, tous les dispositifs et les services qui leur sont associés, doivent avoir une description [uF03]. Donc, il est nécessaire d'associer chaque dispositif (chaque service) à un fichier en *XML* qui le décrit [Law02]. La description d'un dispositif contient tout ce dont il a besoin pour fonctionner, notamment les liens vers les fichiers et les ressources associés à celui-ci,

mais également, le lien vers le fichier de description des services offerts [cf.B.2].

La création d'un objet dispositif **UPnP**^{TM} doit passer par une instanciation de la classe "*Device*". Une fois activé, le nouveau dispositif est annoncé dans tout le réseau **UPnP**^{TM}, en transmettant automatiquement un message "*ssdp::alive*". L'instance de *Device* agit en même temps comme un serveur, puisqu'elle se met à l'écoute des requêtes en provenance des points de contrôle [cf.B.3]. Quand le dispositif est arrêté, un message "*ssdp::byebye*" est transmis au réseau, indiquant que ce dispositif n'est plus en service.

Chaque dispositif fournit un ou plusieurs services que l'on peut visionner par appel à la méthode "*getServiceList()*" de la classe "*Device*". Un service peut avoir des actions et des variables d'état. La classe "*Service*" contient des méthodes permettant de manipuler ces éléments.

10.2.2 Programmation des points de contrôle

Un point de contrôle **UPnP**^{TM} est une instance de la classe "*ControlPoint*". Si la méthode "*start*" de cette classe est appelée, le point de contrôle est alors actif et il envoie un message *multicast* de découverte, cherchant de nouveaux dispositifs fonctionnels sur le réseau. Il agit également comme serveur répondant aux requêtes des dispositifs qui veulent annoncer leur présence, en leur envoyant des réponses confirmant leur inscription dans la liste des dispositifs actifs du réseau. Il reçoit aussi des messages événementiels qui parviennent de différents dispositifs, comme par exemple lorsqu'un nouveau service vient d'être ajouté à la gamme de services offerts par un dispositif

donné, ou lorsqu'un service existant change d'état. Dans le cas où un dispositif quitte le réseau, le point de contrôle l'enlève de sa liste. Les points de contrôle mettent à jour leurs listes des dispositifs actifs d'une façon permanente en exécutant la méthode "search()" qui permet d'envoyer des messages *broadcast* sur le réseau, ensuite, pour chaque dispositif découvert et enregistré, il y a un mécanisme de souscription aux événements en provenance de celui-ci.

10.3 MISE EN ŒUVRE D'UPNP^{TM} SUR L'OSGI^{TM}

La raison pour laquelle **UPnP**^{TM} a été adopté, est qu'il s'agit d'un protocole dont l'architecture réseautique est ouverte et distribuée. Ainsi, **UPnP**^{TM} peut supporter plusieurs technologies pour le contrôle résidentiel, voire être habile à jouer un rôle de supervision pour cette multitude de protocoles. Le protocole $\widehat{\text{X10}}$ en effet peut ainsi cohabité et être contrôlé et homogénéisé avec d'autres technologies grâce à **UPnP**^{TM}. Une autre raison de ce choix est que **UPnP**^{TM} permet d'être un point de rencontre entre des protocoles de communication distants (comme SIP, H.323 etc) et les autres protocoles qui résident à l'intérieur du domaine résidentiel.

Avant de pouvoir implémenter le protocole **UPnP**^{TM} sur la plateforme **OSGi**^{TM}, il fallait suivre quelques étapes en respectant la spécification d'**UPnP**^{TM}[1]. Le dispositif utilisé dans cette application est une lampe de bureau ordinaire, c'est un appareil qui a été testé et contrôlé dans l'étape précédente avec le protocole $\widehat{\text{X10}}$. Dans cette étape, le but était de maintenir

[1]http ://www.upnp.org/

ce contrôle (toujours via le protocole $\widehat{\underline{X10}}$), mais cette fois-ci, l'appareil serait enregistré et géré avec le protocole **UPnP**^{TM} en passant par $\widehat{\underline{X10}}$.

10.3.1 Description d'un "dispositif UPnP^{TM}" simple : une lampe

Même si une simple lampe de bureau ne supporte pas l'**UPnP**^{TM} directement, elle peut être vue comme dispositif **UPnP**^{TM}. En effet, dans cette application elle est contrôlée par $\widehat{\underline{X10}}$ qui appartient au réseau **UPnP**^{TM}. L'écriture de la description de la lampe, comme dispositif appartenant au réseau **UPnP**^{TM}, constitue une étape très importante pour le contrôle de cette lampe via ce protocole. Le schéma [cf.10.1] montre quelques lignes de la balise "device" dérivée de la description de la lampe.

Le service qu'offre la lampe, décrit principalement, dans la balise *<service>* s'appelle "*light*", son dossier d'inscription, comme l'indique le schéma précédent, existe par choix dans le même dossier de la description de la lampe elle même, dans un dossier nommé "service" qui, sous entendu, pourrait contenir d'autres services liés à la lampe[2]. La description du service "*light*"est plus simple encore, il s'agit de donner une version à ce service, et un ensemble d'actions qu'il peut supporter, à savoir {**"*ON*", "*OFF*"**} que l'on a choisi pour cette application. Le schéma [10.2] donne une idée sur cette description.

[2]La lampe est un cas élémentaire de dispositifs que peut contenir un réseau **UPnP**^{TM} . cependant, il pourrait y avoir des lampes plus sophistiquées qui offrent autres types de services qu'un simple éclairage.

```
 <?xml version="1.0" ?>
.................
.................
- <device>
   <friendlyName>Simple Lamp Device</friendlyName>
   <deviceLocation>bedroom</deviceLocation>
   <!--  l'adresse X10 de la lampe -->
   <x10Address>A1</x10Address>
   <manufacturer>Sylvania</manufacturer>
  -<modelDescription>
   Sylvania Lamp Device
   </modelDescription>
   ......................
   ......................
   <UDN>uuid:bedRoomLampDevice</UDN>
   <UPC>123456789012</UPC>
- <serviceList>
- <service>
<!--  le lien vers la description
du service offert par la lampe-->
-<SCPDURL>
/service/light/light_description.xml
</SCPDURL>
</service>
</serviceList>
 </device>
```

FIG. 10.1 – Description du dispositif lampe

10.3.2 Création et installation du bundle upnp-x10

Comme il a été mentionné auparavant, le *bundle* upnp-x10 permet de gagner un temps considérable dans l'exécution de l'application. Il constitue, en outre, une ressource partagée mise à la disposition pour plusieurs autres *bundles*. Les différents objets qui ont utilisé l'un des deux protocoles **UPnP**^{TM} et $\widehat{\underline{\text{X10}}}$ ou les deux ensemble utilisent ce *bundle*. La lampe, sujet d'application pour ce projet, a été présentée logiquement, comme objet

```
 <?xml version=''1.0'' ?>
.................
.................
- <specVersion>
  <major>1</major>
  <minor>0</minor>
  </specVersion>
- <actionList> <!--liste des actions -->
- <action>
   <name>ON</name>
  </action>
- <action>
   <name>OFF</name>
  </action>
  </actionList>
  .................
.................
```

FIG. 10.2 – Description du service "*light*"

"Device" d'abord, puisqu'il serait déclaré dans le réseau **UPnP**^{TM} , et en même temps comme objet client "*Controller*" qui est sensé communiquer avec le serveur X10 décrit précédemment. L'activation de la lampe a été effectuée d'une manière indirecte, par l'intermédiaire d'un "agent" **UPnP**^{TM} que l'on détaillera le concept dans le prochain chapitre. cet agent a été implémenté dans le *Proxy* SIP afin de traduire les messages SIP en simples commandes X10 délivrées à l'objet lampe pour exécuter l'une des deux actions : "*ON*" et "*OFF*".

Nous venons d'expliquer, en gros, la mise en oeuvre du protocole **UPnP**^{TM} sur la plateforme **OSGi**^{TM}. Dans les prochaines sections nous verrons plus en détails le fonctionnement de l'application dans son ensemble et les différents services fournis.

CHAPITRE 11

MODIFICATIONS APPORTÉES AU PROTOCOLE SIP

SIP a été originellement conçu pour la mise en oeuvre de la téléphonie sur IP. Le principe qui a été adopté dans sa création, est l'établissement de sessions d'appel entre deux points. Cependant, ce principe pourrait être généralisé pour couvrir les connexions non durables, pourvu que, dans ce contexte, la phase d'établissement de session puisse être éliminée et le type de la porteuse soit généralisé [MMT01].

Dans ce chapitre, il sera décrit ce qui a été développé dans ce projet pour satisfaire une partie des exigences liées à l'utilisation de SIP dans un contexte de domotique, en particulier pour communiquer avec les appareils domestiques.

La pile SIP codée en Java, qui a été utilisée dans ce projet, est celle de *NIST* [OR03]. L'implémentation de cette pile qui a été développée et réadaptée pour la domotique est celle de *Siptrex*[1]. Le développement de la pile SIP a touché particulièrement les composantes suivantes :

[1] http ://siptrex.net

URL : pour formuler un identificateur constituant la partie à gauche du symbole '@' dans l'adresse d'un appareil, afin de faciliter sa découverte dans le réseau.

Méthode : une nouvelle méthode "*DO*" a été définie spécifiquement pour interagir avec les NAs, dans le cas des actions de contrôle

Type *MIME* : le protocole DMP (*Device Message Protocole*) [KDGS04] a été utilisé comme type de contenu des messages SIP destinés aux NAs.

Les prochaines sections décrivent plus en détail ces modifications.

11.1 ADRESSAGE DOMESTIQUE

Dans les messages SIP, le contenu des entêtes "*TO :*" et "*FROM :*", sont des adresses encodées en format *URL* (*Universal Resource Locators*). Cependant, elles peuvent être des numéros de téléphone également. Dans le présent projet, et afin de faciliter la découverte et la localisation de l'appareil X10 en particulier, un schéma de nomination spécifique concernant la partie gauche du symbole '@' a été utilisé dans l'adresse de destination (adresse de l'appareil) sous le format "*ID@Domaine*". L'identificateur "**ID**", qui doit être unique dans un espace pouvant contenir jusqu'à 256 modules X10 actifs [cf.Fig.6.2], se compose de deux parties concaténées :

1. L'adresse X10 de l'appareil, pour connaître son emplacement
2. Le nom de l'appareil pour, d'une part, fournir plus de lisibilité à l'usager, et d'autre part, pour faciliter l'identification et la description faites par **UPnP**^{TM} [cf.Fig.10.1].

. Ainsi, dans le cas d'une **lampe** branchée sur le module X10 ayant l'adresse ***A1*** par exemple, son identificateur peut être : "***A1Lamp***"

11.2 Méthode "Do"

Une nouvelle méthode SIP a été développée dans ce projet dans l'objectif de communiquer avec les appareils domestiques. Cette méthode, nommée "**Do**", permet d'étendre SIP pour le rendre habile à supporter d'autres types de charges utiles (*Payloads*) que le SDP (*Session Description Protocol*). Tout type *MIME* prédéfini, pourrait être utilisé comme charge utile d'un contenu SIP. En outre, de nouveaux types *MIME* ultérieurement définis pour des classes particulières d'appareils, pourraient être supportés. **Do** permet de transmettre la commande appropriée pour un appareil cible, tel qu'une commande *ON* pour une lampe par exemple. Une fois envoyée, la commande déclenche une réponse simple, indiquant son résultat, qui est naturellement transmise par les mécanismes standards de réponse de SIP.

Avant d'avoir pu utiliser cette méthode dans les messages SIP, il a fallu effectuer un certain nombre de modifications sur la pile SIP elle-même. Le tableau [reftab :comparMethodes] décrit la différence entre les deux méthodes, *INVITE* et *DO*

11.3 Contenu des messages : Protocole DMP

Typiquement, la méthode ***INVITE*** de SIP utilise le protocole SDP comme type *MIME* pour l'envoi des messages. Dans un contexte de la domotique, il est préférable d'utiliser un nouveau type *MIME* spécifique pour communiquer avec les dispositifs. Le protocole ***DMP***, qui a été défini par le groupe de recherche de *Telcordia* [KDGS04], a été implémenté dans ce projet. Ce protocole, dont la spécification est basée sur XML, est semblable au protocole **UPnP**^{TM}[2], et il permet de transporter les commandes encapsulées dans des balises, selon des schémas prédéfinis. Ses principaux avantages ce sont son extensibilité et sa flexibilité. En effet, dans un domaine résidentiel, il est tout à fait raisonnable d'avoir une multitude d'appareils appartenant à différents fabricants, et ayant différentes fonctionnalités. Le DMP est apte à prendre en charge tous ces appareils, puisqu'il fonctionne indépendamment des types variés.

Les commandes qui peuvent être injectées dans un contenu de type DMP sont de trois catégories :

Méthode classique : "INVITE"	**Nouvelle méthode : "DO"**
INVITE sip :bob@acme.com SIP/2.0	DO sip :A1Lamp@myhome.net SIP/2.0
To : Bob B. <sip :bob@acme.com>	To : sip :A1Lamp@myhome.net
Content-Type : application/SDP v=0 o=Alice 55765 687637 IN IP4 128.3.4.5 s=Call from Alice c=IN IP4 alice_ws.radvision.com M=audio 3456 RTP/AVP 0 3 4 5	Content-Type : application/DMP <?xml version=1.0?> <DMPAction> <Device>A1Lamp</Device> <Action>ON</Action> </DMPAction>

TAB. 11.1 – Comparaison entre les méthodes : *INVITE* et *DO*

[2]D'ailleurs ça fait partie des raisons pour lesquelles l'**UPnP**^{TM} a été utilisé dans ce projet

1. **Commandes "Requêtes" :** permettent de demander la valeur d'une variable d'état relative à un ou plusieurs dispositifs, par exemple, la température ambiante à l'intérieur d'un habitat
2. **Commandes "Événement" :** permettent de demander à un dispositif de communiquer la valeur de sa variable d'état lorsqu'un événement prévu se manifeste
3. **Commandes "Contrôle" :** permettent de demander à un dispositif de changer son état, comme la commande "ON" envoyée à une lampe.

C'est la dernière catégorie qui a été utilisée dans la démonstration de ce projet, le tableau [cf. 11.1] montre un exemple de contenu DMP d'un message SIP quand la méthode DO est utilisée [KDGS04].

CHAPITRE 12

INTÉGRATION AU SEIN D'UNE APPLICATION TEST

L'objectif de cette phase du projet est de réaliser une application de démonstration. Cette application regroupe les différentes technologies vues dans ce mémoire, à savoir, SIP, **UPnP**^{TM} et $\widehat{\underline{X10}}$, installées sur la plateforme **OSGi**^{TM}. Le but est de pouvoir contrôler un réseau domestique via Internet en utilisant SIP comme protocole de communication. Un usager SIP peut contrôler les appareils domestiques de son appartement depuis son bureau distant [cf.Fig.12.1]. L'application de démonstration est composée de deux parties majeures, interne et externe.

12.1 COMMUNICATION À L'INTÉRIEUR DE L'HABITATION

La partie interne de l'habitation comporte la passerelle de services et le réseau domestique. C'est donc la partie qui a occupé la majorité du temps de

travail pour l'intégration des technologies citées plus haut dans ce mémoire, à savoir, la plateforme **OSGi**^{TM} (hébergeante) et les protocoles $\widehat{\underline{X10}}$, **UPnP**^{TM} et SIP sous formes de *bundles* [cf.12.1]. Pour chaque technologie, il a fallu effectuer une étude théorique, choisir l'*API* Java la plus adéquate par rapport à la spécification **OSGi**^{TM}et la tester, et finalement la rectifier pour l'intégrer sur la plateforme **OSGi**^{TM}.

En ce qui concerne le réseau de tests internes, une lampe de bureau est branchée à un module X10 (*AM12*) tandis qu'un module contrôleur (*CM11A*) le gère. Le module contrôleur est branché au port série d'un PC. Les deux modules sont branchés également au courant électrique et formant ainsi un réseau $\widehat{\underline{X10}}$. En termes de services, un objet UPnP s'exécute auprès du Proxy SIP en l'informant qu'il y a un dispositif de type lampe branché au réseau et prêt à recevoir des commandes.

12.2 COMMUNICATION EXTERNE AVEC L'HABITATION

La partie externe de la démonstration consiste à développer les composantes de l'agent SIP qui va communiquer avec la lampe. Cet agent exploite la majorité des extensions apportées à la pile SIP [cf.chap.11], puisque tous les messages utilisant la méthode DO en proviennent.

12.3 SCÉNARIO DE LA COMMUNICATION AVEC LA LAMPE

Le scénario de la communication entre l'agent et la lampe est le suivant :

- Après l'identification de l'utilisateur, le proxy lui envoie la liste[1] des appareils enregistrés sur le réseau. En sélectionnant un de la liste (la lampe), l'adresse SIP de cet appareil s'affiche dans la zone adresse de destination[2] [cf.sect.11.1]. Ensuite, l'utilisateur choisit l'action qu'il désire envoyer à la lampe (*ON* ou *OFF*), et appuie sur le bouton *DO*.
- Le proxy SIP est, par définition, un agent SIP particulier. Donc, il a été choisi dans ce projet pour jouer deux rôles en même temps :

  1 le rôle de proxy naturel pour le contrôle des communications

  2 celui d'un agent SIP qui reçoit les messages en provenance de l'agent externe.

  Lorsque l'agent SIP envoie le message DO, le proxy lui envoie un message d'accusé de réception normal, ensuite il délivre le contenu du message à l'objet UPnP, mentionné plus haut, qui a une adresse SIP fixe et prédéfinie dans le programme.
- L'agent UPnP communique la commande au contrôleur X10. Celui-ci ayant été en écoute, il redirige la commande à la lampe désignée par son adresse X10.

[1]Pour cette démonstration, la liste contient un seul élément qui est la lampe. Une ligne de la liste cotient le nom de l'appareil plus son adresse X10.

[2]En connaissant la partie à droite du signe '@', la reconstitution de la partie à gauche se fait par la concaténation de l'adresse de la lampe et de son adresse X10.

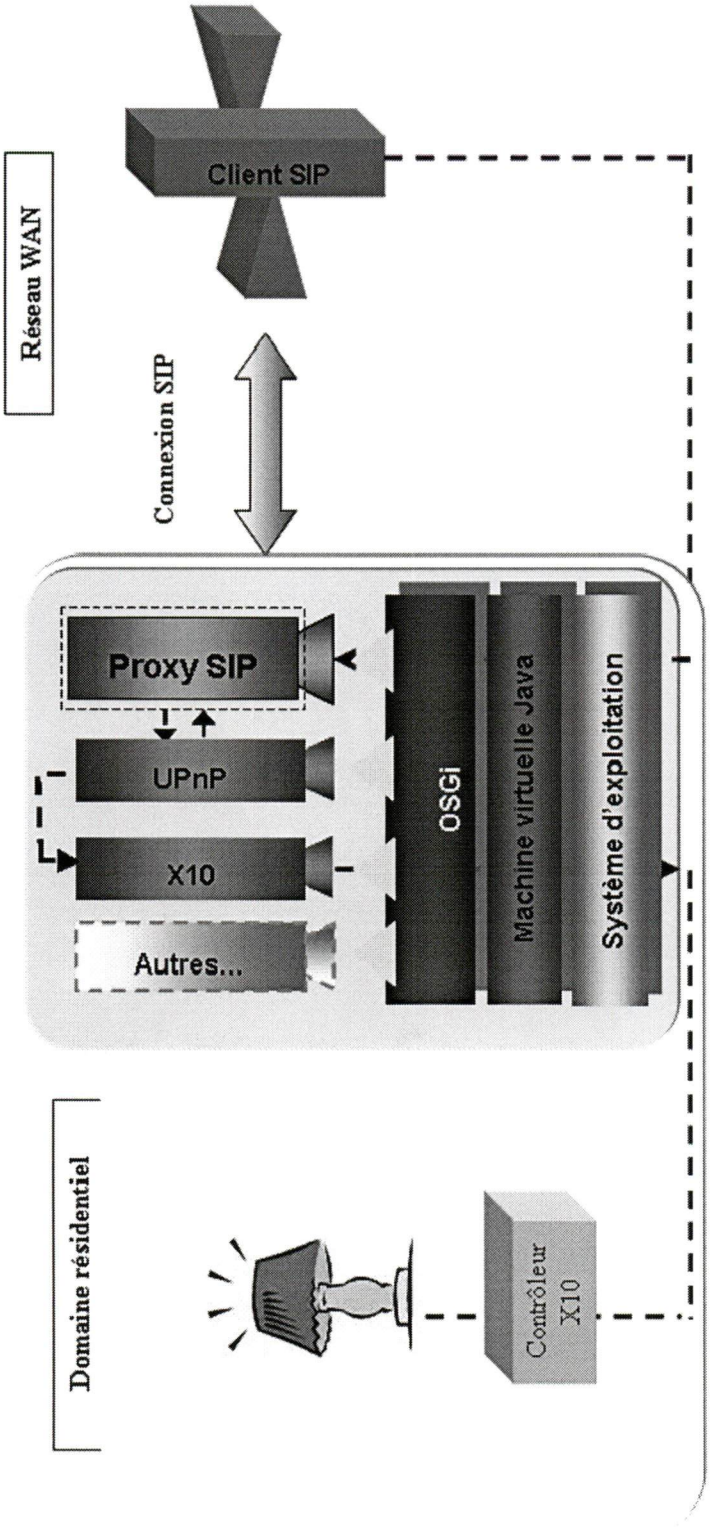


FIG. 12.1 – Intégration de différents services pour communiquer avec les appareils domestiques distants.

CHAPITRE 13

CONCLUSION

Le présent mémoire décrit une architecture et son implémentation permettant de contrôler à distance des appareils domotiques, par l'intégration de différentes technologies. À l'intérieur de l'habitat, le protocole **UPnP**^{TM} permet de gérer les appareils domestiques indépendamment de leurs caractéristiques matérielles et des services qu'ils peuvent fournir. L'architecture d'**UPnP**^{TM} se caractérise par sa flexibilité et son extensibilité. Le protocole SIP (*Session Initiation Protocol*) conçu originellement pour la signalisation et la téléphonie sur IP a été utilisé dans ce projet afin de communiquer avec les appareils domestiques depuis un endroit externe de l'habitat via la passerelle résidentielle **OSGi**^{TM}. Cette passerelle permet d'héberger les services sous forme de *bundles* .

La preuve de concept implémentée dans ce projet consiste à commander une lampe de bureau à distance. Le service X10 qui gère cette lampe a été implémenté sur la passerelle **OSGi**^{TM} ainsi que le protocole **UPnP**^{TM} et le Proxy SIP. Ce sont donc les services constituant les technologies de contrôle domestique qui ont été implémentées à l'intérieur de l'habitat. Pour contrôler la lampe depuis un endroit lointain, le protocole SIP a été utilisé.

Plusieurs modifications ont été apportées à ce protocole de signalisation afin de l'adapter au domaine résidentiel pour communiquer avec les appareils domestiques. La principale extension de SIP, est l'introduction d'une méthode "DO" qui permet d'envoyer des messages de commandes aux appareils domestiques. L'exécution de cette méthode n'exige pas l'établissement d'une véritable session SIP, comme dans le cas d'échange multimédia avec le protocole RTP, puisque le type MIME utilisé dans cette méthode est le DMP (*Device Message Protocole*), un protocole basé sur XML, permettant d'encapsuler les commandes envoyées aux appareils dans des balises spécifiques. C'était donc la deuxième modification qui a été apportée à la pile SIP dans ce projet [cf.sect.11.3]. Avec ces modifications, une communication a été établie avec la lampe depuis un agent SIP distant. Cet agent envoie des commandes "ON" et "OFF" au proxy SIP s'exécutant sur la passerelle, afin que ce dernier les délivre au service UPnP qui contrôle l'habitat en interne. Le service utilise ensuite X10 pour acheminer le message à la lampe.

Il y a plusieurs travaux possibles qui pourraient être ajoutés à ce sujet. L'adressage qui a été utilisé dans cette méthode, malgré qu'il a subit quelques modifications, reste un adressage de SIP classique (sip:ID@domaine.com). Cependant, pour un contexte domestique, et pour plusieurs facteurs qui y sont liés, un adressage spécifique, notamment dans l'entête "TO", s'avère être mieux approprié pour garder la vie privée de l'utilisateur final loin d'être dévoilée par des usagers malintentionnés, surtout si cet utilisateur préfère garder le contenu de son habitat et le type d'appareils qu'il utilise, à l'écart des yeux étrangers. Il serait plus sûr encore

de chiffrer la partie à gauche du caractère '@' dans l'adresse de destination, qui devrait en général contenir le nom de l'appareil et son emplacement dans l'habitat.

Dans ce projet, le type de communication qui a été effectuée avec la lampe est unidirectionnel. L'utilisateur envoie la commande "ON" à la lampe pour s'allumer, mais par contre, il ne reçoit aucun indice lui permettant de vérifier l'état réel de la lampe. Dans des cas plus critiques, des contrôleur devraient pouvoir envoyer à l'utilisateur des échos à ses commandes, ou bien des messages spontanés asynchrones. Il serait donc plus consistant d'ajouter d'autres méthodes au protocole SIP qui permettent aux différents contrôleurs et appareils intelligents, d'envoyer des réponses à l'utilisateurs ou de le notifier si un événement se manifeste dans son habitat. À titre d'exemple, l'ajout d'une méthode nommée "Notify" à SIP permet de rendre la communication bidirectionnelle avec les appareils. Ainsi, un caméra de surveillance supportant SIP peut utiliser la méthode 'Notify" pour aviser l'utilisateur distant d'un éventuel visiteur. Ce caméra n'a qu'à envoyer un message au proxy SIP installé sur la passerelle résidentielle. Le proxy se charge, par la suite, de le rediriger vers l'utilisateur. Celui-ci reçoit ainsi un message textuel sur son téléphone SIP lui demandant, également, s'il voudrait visionner les images ayant été prises par son caméra.

Quatrième partie

Acronymes, annexes et bibliographie

ACRONYMES ET ABRÉVIATIONS

DMP Device Message Protocole

GENA General Event Notification Architecture

ITU International Telecommunication Union

LAN Local Area Network

NAT Network Address Translation

OSGi Open Service Gateway Initiative

RSVP Resource Reservation Protocol

RTCP Real-Time streaming Control Protocol

RTP Real-time Transport Protocol

SAP Session Announcement Protocol

SDP Session Description Protocol

URL Universal Resource Locators

VoIP Voice over Internet Protocol

WAN Wide Area Network

ANNEXE A

Bundle "X10Server"

A.1 CLASSE *Activator*

```
package upnp.x10.server;
import javax.comm.CommDriver;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import x10.*;
import x10.net.*;

public class Activator implements BundleActivator{

public void start(BundleContext bc) throws Exception {
String serialPort="COM2";  //ici le port utilisé est fixe
ControllerServer cServer;
......
cServer = new ControllerServer(controller, 2500);
cServer.start();
  }

  public void stop(BundleContext context) throws Exception {
   ......
  }
}
```

Classe *Activator* du *bundle X10Server*

A.2 CONTENU DU *bundle X10Server*

```
Bundle-Name: X10Server
Manifest-Version: 1.0
Bundle-Description: X10 application Server
Bundle-Version: 1.0
Bundle-Activator: upnp.x10.server.Activator
Bundle-ContactAddress: a.chtatou@USherbrooke.ca
Bundle-ClassPath: .,commwin32.jar,upnp-x10.jar
Bundle-NativeCode: upnp/x10/server/win32com.dll;
processor=x86; osname=Windows XP
```

Fichier *Manifest* du *bundle X10Server*

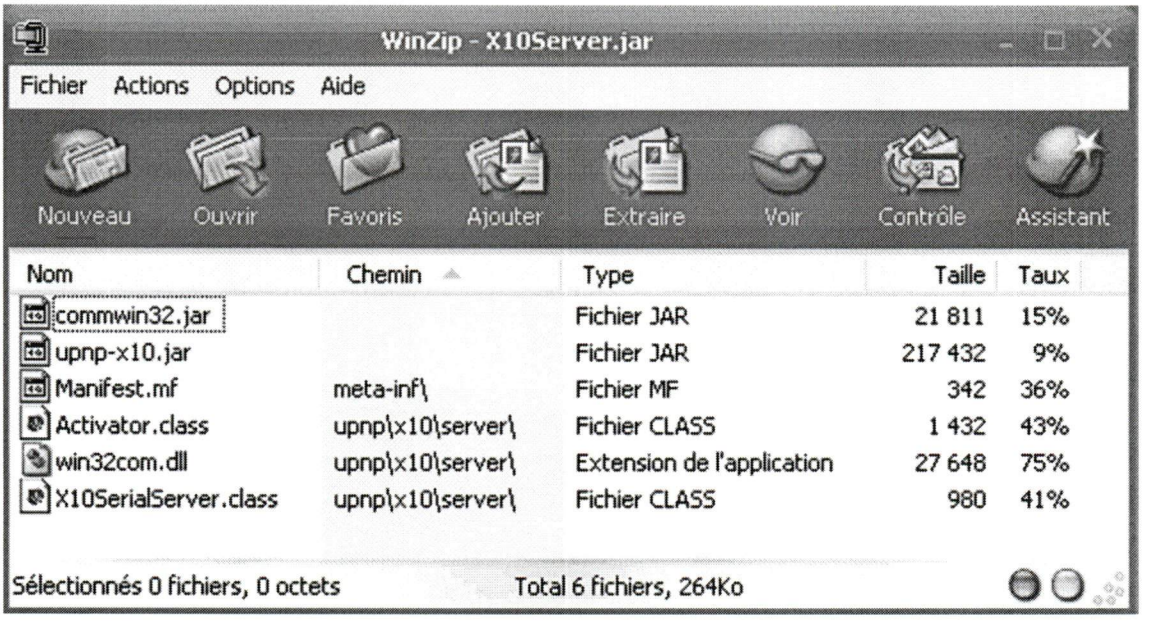


FIG. A.1 – Contenu du fichier X10Server.jar

ANNEXE B

CODAGE ET DESCRIPTION DES COMPOSANTES **UPNP**^{TM}

B.1 DIAGRAMME DES CLASSES POUR LES DISPOSITIFS UPNP^{TM}

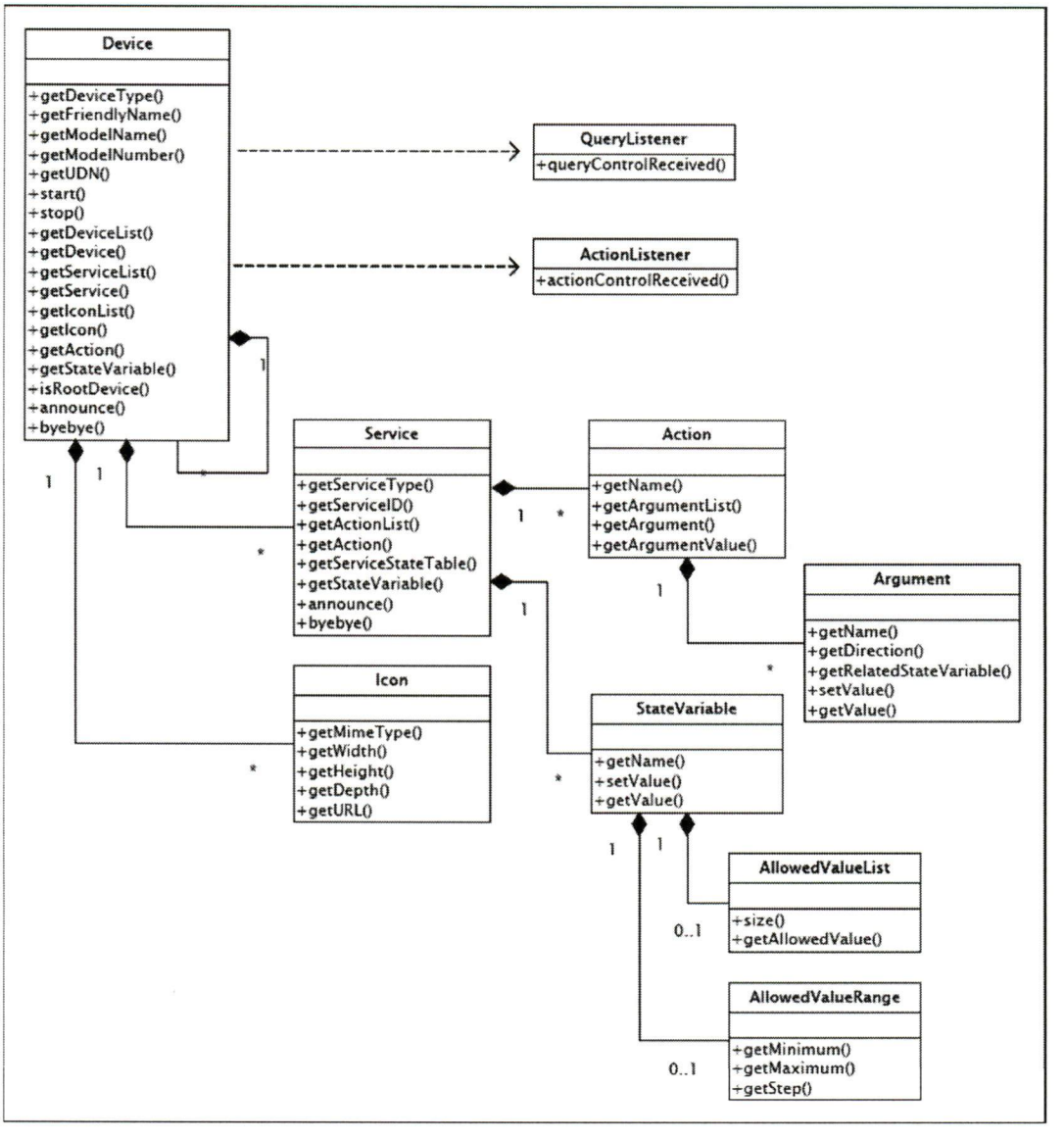


FIG. B.1 – Diagramme UML de la partie dispositifs de *Cyberlink*

B.2 DESCRIPTION DES DISPOSITIFS ET DES SERVICES ASSOCIÉS

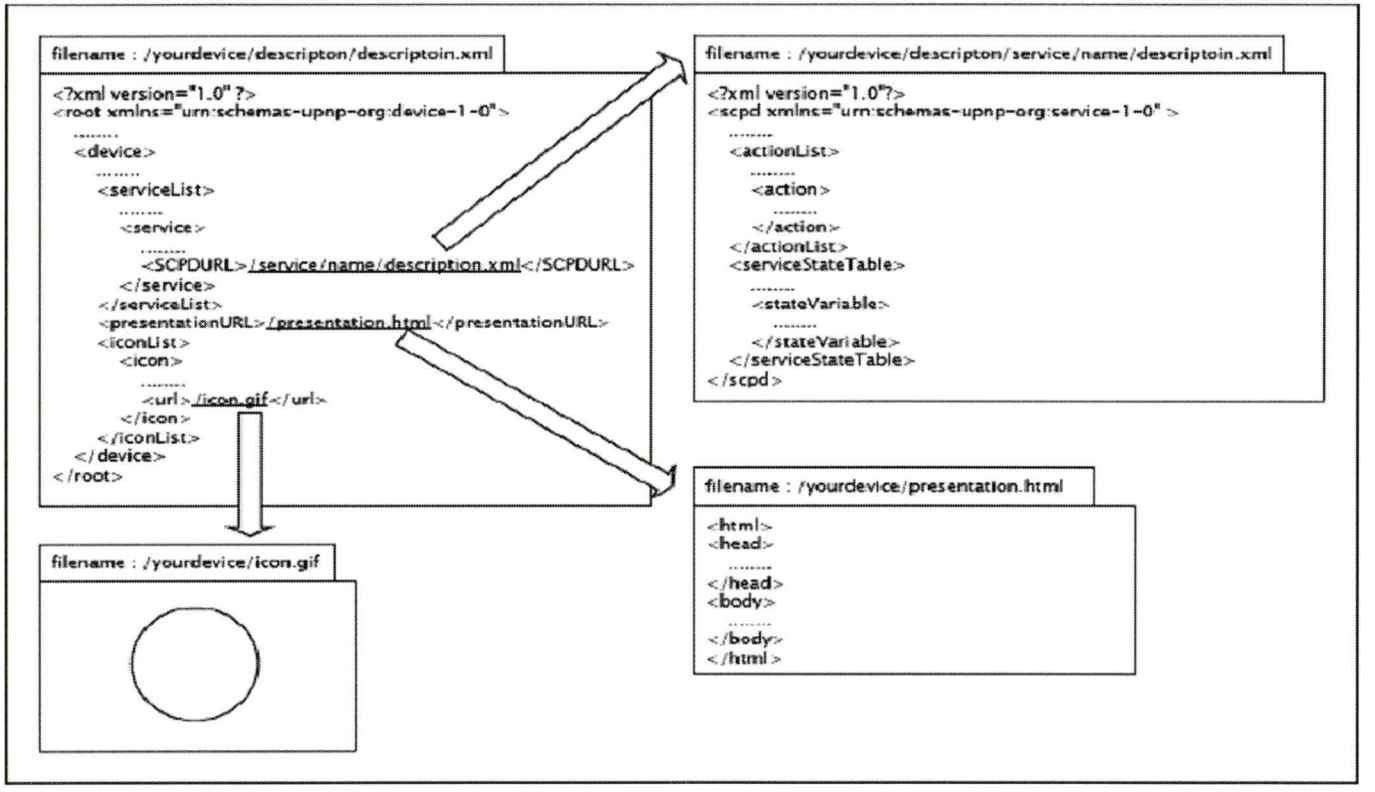


FIG. B.2 – Exemple de fichiers et ressources liés à la description d'un dispositif

Le schéma montre que le fichier de description lié au dispositif "*yourdevice*", fait référence à d'autres fichiers et ressources, en indiquant des liens relativement locaux vers eux.

B.3 NOTIFICATION POUR UN LANCEMENT DE DISPOSITIF

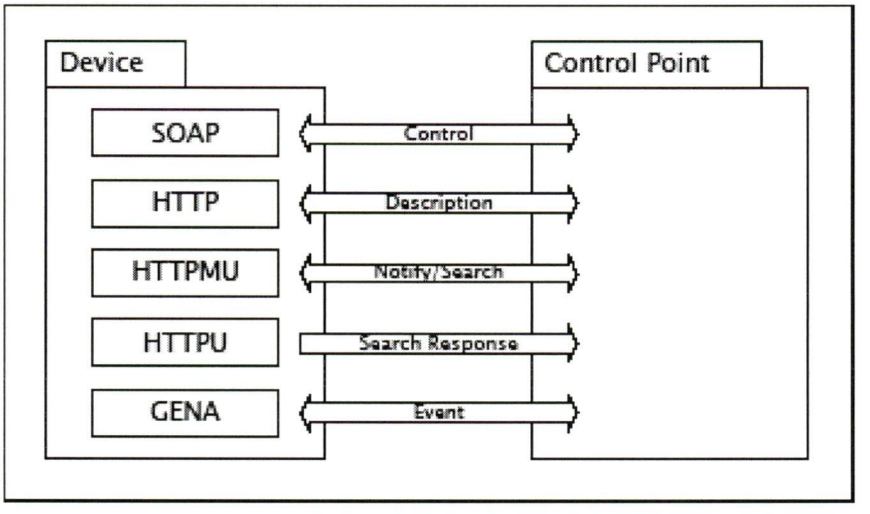


FIG. B.3 – Echange de messages entre un dispositif actif et un point de contrôle

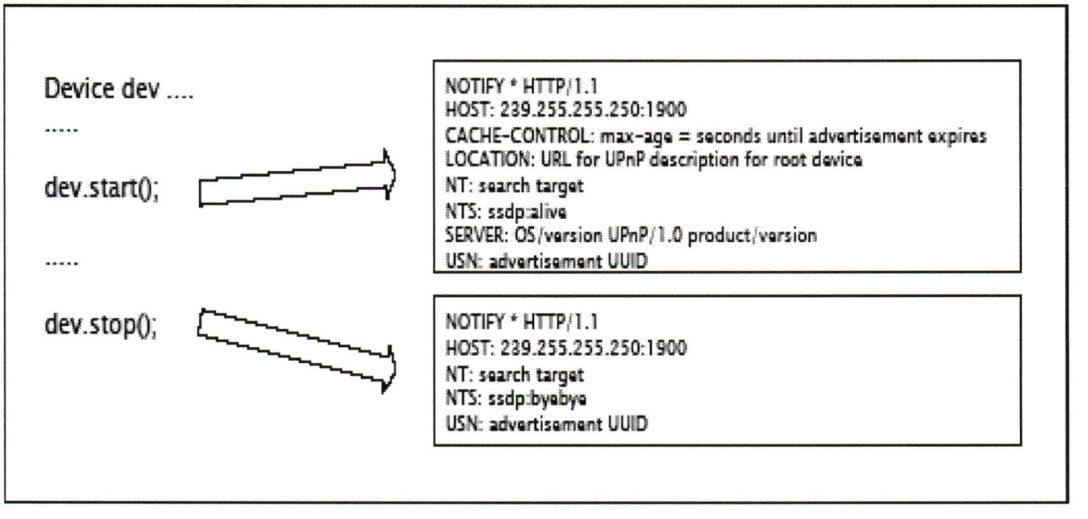


FIG. B.4 – Messages de notification de deux états différents d'un dispositif

B.4 DIAGRAMME DES CLASSES POUR LES POINTS DE CONTRÔLE

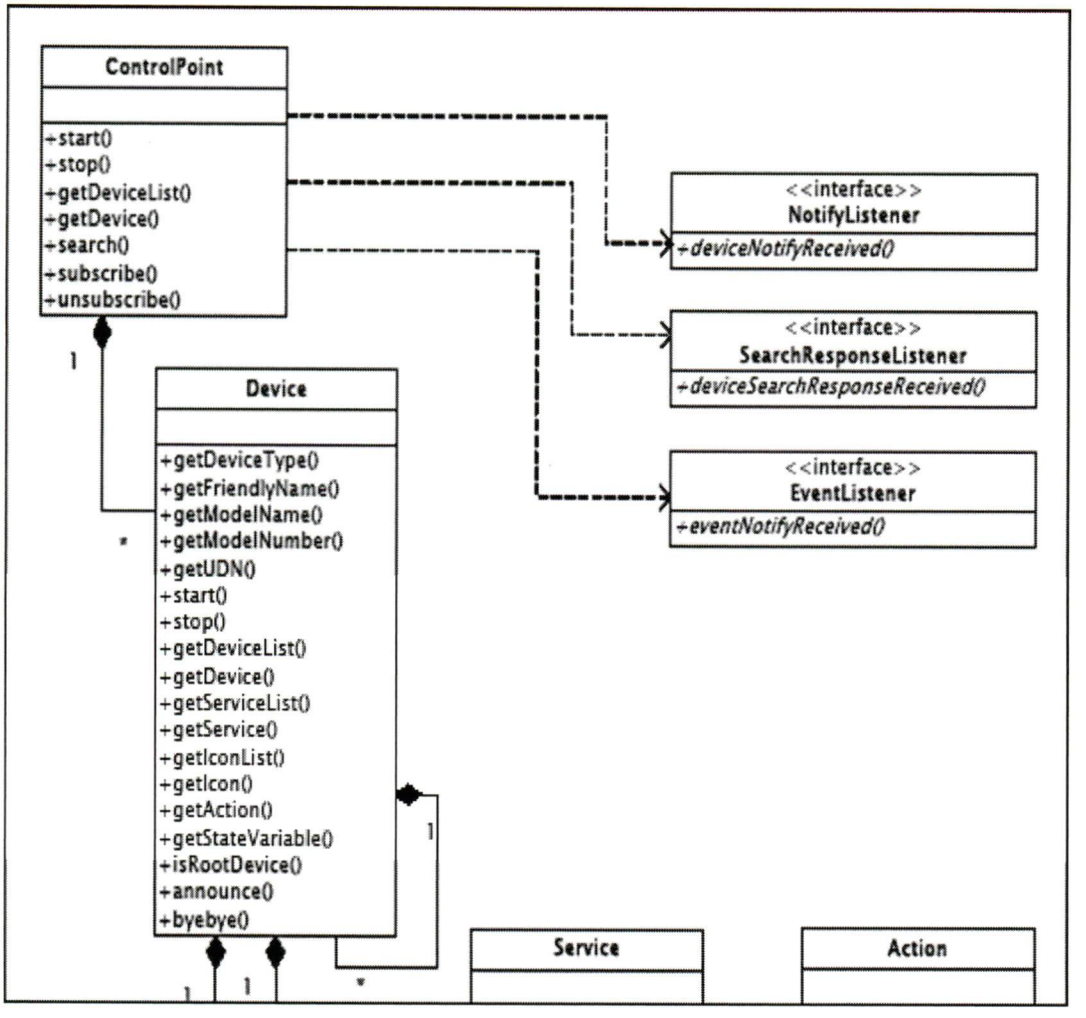


FIG. B.5 – Diagramme UML de la partie points de contrôle de *Cyberlink*

Un point de contrôle est considéré comme un type particulier d'un dispositif, ce qui explique l'héritage de la classe "*ControlPoint*" de la "*classe Device*"

ANNEXE C

INTERFACE UTILISATEUR SIP

C.1 EXÉCUTION DE LA COMMANDE DO

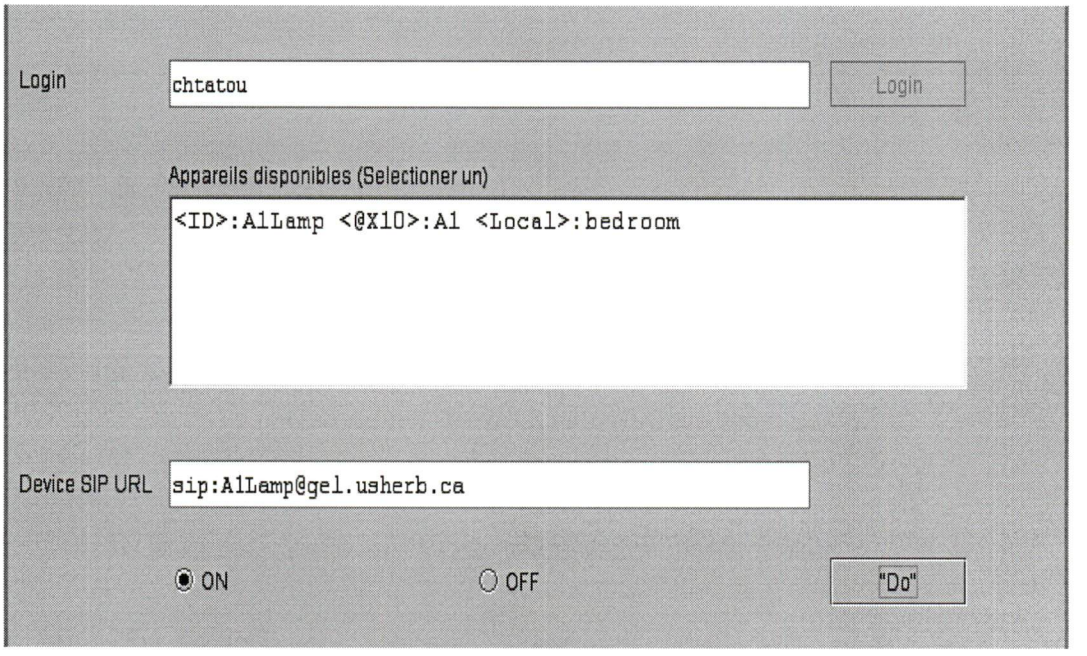


FIG. C.1 – Envoi de la commande DO à une lampe

C.2 CONTENU DU MESSAGE DO

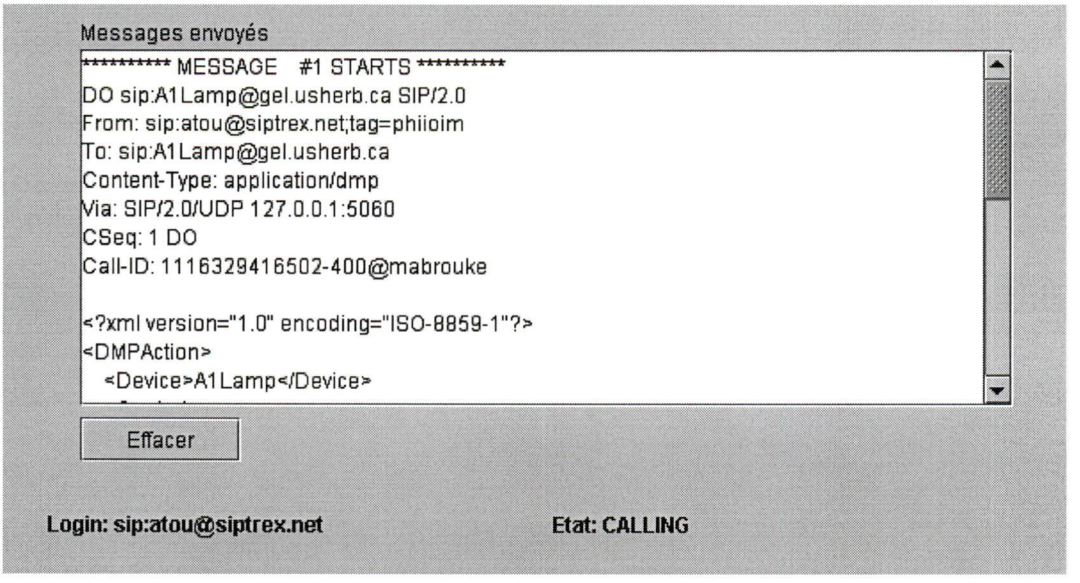


FIG. C.2 – Message envoyé au Proxy SIP

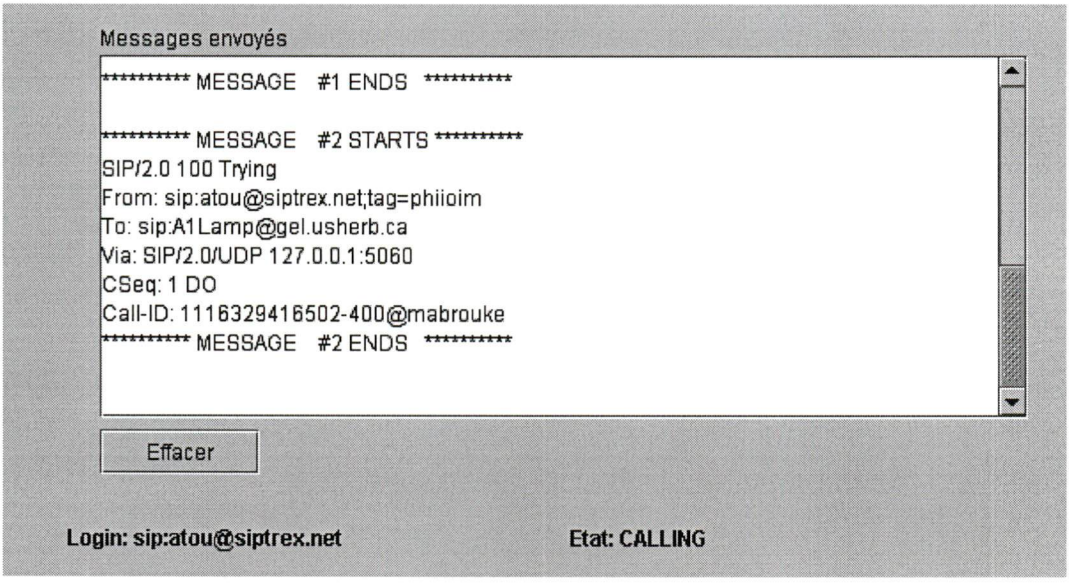


FIG. C.3 – Message de réponse du Proxy SIP

BIBLIOGRAPHIE

[Dat98] DATABEAM© – *A primer on the h.323 series standard*, DataBeam, DataBeam Corporation 3191 Nicholasville Road Lexington, Kentucky 40503 USA, version 2.0 éd., May 1998, This document may be reproduced, provided such reproduction is performed in its complete, unaltered form.

[GC01] L. GONG et K. CHEN – *Programming open service gateways with java embedded server^{TM} technology*, 1re éd., Addison-Wesley; ISBN : 0201711028, 2001, 480 pages.

[Ide03] X. IDEAS – « The Basics of X-10 », Tech. report, 2003.

[KDGS04] S. KHURANA, A. DUTTA, P. GURUNG et H. SCHULZRINNE – « XML based Wide Area Communication with Networked Appliances », Tech. report, 2004.

[Kin99] P. KINGERY – « Digital X-10 », *Home Toys* (1999).

[Law02] S. LAWRENCE – « Basic Device Definition, version 1.0 », Tech. report, GlobespanVirata, 2002, UPnP Device Description.

[McL01] B. MCLAUGHLIN – *Java and xml*, 2eme éd., O'Reilly, 2001, Solutions to Real-World Problems.

[MMT01] S. MOYER, D. MAPLES et S. TSANG – « A Protocol for Wide-Area Secure Networked Appliance Communication », *Communication Magazine, IEEE Communication Society, New York* (2001).

[MMTG02] S. MOYER, D. MAPLES, S. TSANG et A. GHOSH – « Service Portability of Networked Appliances », *Communication Magazine, IEEE Communication Society, New York* (2002), p. 116–121.

[Moe02] P. MOERMANS – *Implementation of a Home-Server Using the OSGi Architecture*, Master thesis, Faculté des Sciences appliquées, Université Libre de Bruxelles, Belgique, 2002, Travail de fin d'études d'Ingénieur Civil Informaticien.

[oA03] **OSGi**^{TM} ALLIANCE – *Osgi service platform specification*, IOS Press, The Netherlands, Version 3.0, 2003.

[OR03] P. O'DOHERTY et M. RANGANATHAN – « JAIN SIP Tutorial, Serving the Developer Community », Tech. report, Sun Microsystems, National Institut of Standards and Technology, 2003.

[RSC+02] J. ROSENBERG, H. SCHULZRINNE, G. CAMARILLO, A. JOHNSTON, J. PETERSON, R. SPARKS, M. HANDLEY et E. SCHOOLER – « SIP : Session Initiation Protocol », RFC 3261, IETF, 2002.

[Sat02] K. SATOSHI – « CyberLink for Java, Developpement Package for UPnP Devices », Tech. report, Cyber Garage, 2002.

[SC04] R. S.HALL et H. CERVANTES – « An OSGi Implementation and Experience Report », *IEEE Consumer Communications & Networking Conference (CCNC), Las Vegas, USA*, 2004.

[SUN00a] SUN© – « How to Write Your First JES Service », Tech. report, Java Embedded Server^{TM}Version 2.0, 2000.

[SUN00b] — , « Java Embedded Server^{TM} Developer Guide », Tech. report, Java Embedded Server^{TM}Version 2.0, 2000.

[TMM04] K. J. TURNER, E. H. MAGILL et D. J. MARPLES – *Service Provision : Technologies for Next Generation Communications*, Wiley, ISBN 0-470-85066-3, 2004, Hardcover - 386 pages.

[uF03] **UPnP**^{TM} FORUM – « **UPnP**^{TM} Device Architecture », Tech. report, 2003.

[VF02] D. VALTCHEV et I. FRANKOV – « Prosyst Software AG : Service Gateway Architecture for a Smart Home », *IEEE Communications Magazine* **40** (2002), p. 126 –132.

[Was03] W. WASSENBERG – « The java X10 Project », Tech. report, GNU Homelinux, 2003.